Encoding the Sensor Allocation Problem for Reinforcement Learning

Dylan R. Penn

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Aerospace Engineering

Jonathan T. Black, Chair
Kevin K. Schroeder
Michael Fowler
Leonard A. Smith

April 18, 2024
Blacksburg, Virginia

Keywords: space traffic management, resource allocation, reinforcement learning

Encoding the Sensor Allocation Problem for Reinforcement Learning

Dylan R. Penn

(ABSTRACT)

Traditionally, space situational awareness (SSA) sensor networks have relied on dynamic programming theory to generate tasking plans which govern how sensors are allocated to observe resident space objects. Deep reinforcement learning (DRL) techniques, with their ability to be trained on simulated environments, which are readily available for the SSA sensor allocation problem, and demonstrated performance in other fields, have potential to exceed performance of deterministic methods. The research presented in this dissertation develops techniques for encoding an SSA environment model to apply DRL to the sensor allocation problem. This dissertation is the compilation of two separate but related studies. The first study compares two alternative invalid action handling techniques, penalization and masking. The second study examines the performance of policies that have forecast state knowledge incorporated in the observation space.

Encoding the Sensor Allocation Problem for Reinforcement Learning

Dylan R. Penn

(GENERAL AUDIENCE ABSTRACT)

Resident space objects (RSOs) are typically tracked by ground-based sensors (telescopes and radar). Determining how to allocate sensors to RSOs is a complex problem traditionally performed by dynamic programming techniques. Deep reinforcement learning (DRL), a subset of machine learning, has demonstrated performance in other fields, and has the potential to exceed performance of traditional techniques. The research presented in this dissertation develops techniques for encoding a space situational awareness environment model to apply DRL to the sensor allocation problem. This dissertation is the compilation of two separate but related studies. The first study compares two alternative invalid action handling techniques, penalization and masking. The second study examines the performance of policies that have forecast state knowledge incorporated in the observation space.

Dedication

To Lina,

Your ceaseless support has sustained me over the past three years,

You are the best partner, editor, head coach, and cheerleader I could ask for,

This effort was only possible with you by my side.

To coffee,

Without which, this work would also not have been possible.

Acknowledgments

As the proverb goes, “it takes a village to raise a grad student,” or something to that effect. There is a whole village of students, staff, faculty, and friends who helped me conduct the research in this dissertation. It is fruitless to try to name them all, so I’ve just called out those that had a very specific hand in my journey back to school.

To Dr. Black, thank you for bringing me out to Virginia Tech and the SSA lab, and for all your support in the writing process. Your always-positive words have been a great source of encouragement, particularly when my results were not what I wanted. Dr. Schroeder, thank you for introducing me to the sensor allocation topic and getting me started down the incredibly interesting and intellectually rewarding subject of multi-stage decision processes. Dr. Fowler, thank you for your guidance in all things machine learning; every thirty-minute discussion with you boosted my knowledge a month, which was invaluable on a tight timeline. Dr. Smith, thank you for your sage wisdom to focus on what is “adequate for purpose,” which inspired the custody metric work. Collin, thank you for your tutelage and patience in teaching me how to use Ray. To my lab-mates, Dylan, Jon, Cameron, Amit, Connor, and Ginny, thank you all for your comradery, support, and generally superb vibes. You all make coming to work fun. Oesa, thank you for lobbying on my behalf to convince the bosses to send me on a three year sabbatical to dive deep again. I would not be here without your support.

Contents

List of Figures

List of Tables

List of Symbols

Operators

log	Logarithm function to base 10
$\|\cdot\|$	Magnitude of vector
$\|\cdot\|$	Determinant of matrix
$\|\cdot\|$	Number of elements in a set
$\mathbb{E}$	Expectation operator
$\odot$	Hadamard product (element-wise matrix multiplication)
σ	Sigmoid function
tr	Trace of matrix
N	Normal distribution
U	Uniform distribution

Diacritics

$\bar{\bar{\cdot}}$	Mean of means
$\bar{\cdot}$	Mean
$\hat{\cdot}$	Estimate

Variables

$0_{a\times b}$	Zero matrix of shape $[a, b]$
$\bar{P}$	Mean of collection of covariance traces
$\Delta\bar{R}$	Difference in mean returns between final and first training iterations
γ	Discount factor
$\mathbb{N}_0$	The set of natural numbers, including 0
$\mathbb{R}$	The set of real numbers
$\mathcal{M}$	Set of sensors in network
$\mathcal{N}$	Set of resident space objects in catalog
$\mathcal{N}^m_{(\cdot)}$	Set of resident space objects available to mth sensor, subject to constraint $(\cdot)$
$\mathcal{N}_{(\cdot)}$	Set of resident space objects in catalog, subject to constraint $(\cdot)$
$\mathcal{O}$	Observation
Ω_{c}	Continuous visibility map
Ω_{d}	Discrete visibility map
Φ	Action mask
ρ	Custody threshold
σ	Standard deviation
$\varepsilon_{\mathrm{aa}}$	Active action subsidy (> 0)
ε_{l}	Logit override parameter $(\ll 0)$

ε_{na} Null action subsidy

ε_{nv} Non-visible action penalty (> 0)

a Action array

b_f, b_i, b_o Bias vectors in a long short-term memory cell

C Custody fraction

c Custody status of targets

C^1 Continuously differentiable function

c_p Custody proximity of targets

C_t Cell state

F_{fo} Failed observation fraction

F_{ia} Invalid action fraction

F_{na} Null action fraction

F_{nt} Non-trivial opportunity fraction

F_{wo} Wasted opportunity fraction

f_t Forget gate

h_t Hidden state

I_n Identity matrix of size n

i_t Input gate

k, x_0 Sigmoid function parameters

l	Logits array
M	Number of sensors
m	Mean
N	Number of resident space objects
N_{ep}	Number of episodes
N_{nt}	Number of non-trivial opportunities
o_t	Output gate
P	Covariance matrix (km^2, m^2/s^2)
P^-	A priori covariance matrix (km^2, m^2/s^2)
P^n	Covariance matrix of target n (km^2, m^2/s^2)
P_0	Initial covariance matrix (km^2, m^2/s^2)
$P_{1:3}$	Positional covariance matrix (km^2)
Q	Process noise matrix
R	Measurement noise matrix
R	Return (cumulative reward) from an episode
r	Position vector magnitude
r	Reward from a single step
R_{E}	Radius of Earth (6378.1363 km)
s	Generic state

s	Standard deviation
s	Tasking staleness (s)
T	Number of time steps in episode (horizon)
t	Time step index
U	Trajectory of times remaining until next access window for all targets
u	Times remaining until next access window for all targets
v	Visibility value between a sensor and target
W	Trajectory of number of access windows remaining for all targets
w	Number of access windows remaining for all targets
W_f, W_i, W_o	Weighting matrices in a long short-term memory cell
x	Dynamic state vector
x	Generic variable
z	Measured state

Acronyms

A2C advantage actor-critic

A3C asynchronous advantage actor-critic

AC actor-critic

AI artificial intelligence

AL argument of latitude

ASAT anti-satellite weapon

CNN convolutional neural network

COTS commercial off-the-shelf

DDQN double deep Q network

DL deep learning

DQN deep Q learning

DRL deep reinforcement learning

ECI Earth-centered inertial

EKF extended Kalman filter

EO electro optical

FC fully-connected

FIG Fisher Information Gain

FOR field of regard

FOV field of view

GEO geostationary Earth orbit

GSO geosynchronous Earth orbit

IQR inter-quartile range

JGM-3 Joint Earth Gravity Model 3

KF Kalman filter

KL Kullback–Leibler

LEO low Earth orbit

LIDAR light detection and ranging

LSTM long short-term memory

MDP Markov decision process

MEO medium Earth orbit

ML machine learning

NGO non-governmental organization

NN neural network

PPO proximal policy optimization

RAAN right ascension of the ascending node

ReLU rectified linear unit

RF radio frequency

RL reinforcement learning

RNN recurrent neural network

RSO resident space object

SGD stochastic gradient descent

SGP4 Simplified General Perturbations 4

SIG Shannon Information Gain

SMA semi-major axis

SNR signal to noise ratio

SSA space situational awareness

SSN Space Surveillance Network

STD standard deviation

UKF unscented Kalman filter

Chapter 1

Introduction

Since the dawn of the Space Age, satellites and their associated debris have been tracked by networks of sensors. The last roughly two decades have experienced a significant increase in the number and growth rate of new resident space objects (RSOs), the term encompassing all satellites and debris [1, 2]. Looking forward, the rise of mega-constellations and kinetic anti-satellite weapons (ASATs), which produce thousands of detectable debris fragments, will only accelerate the population growth of objects in Earth orbit. As the RSO population increases, the probability of collisions increases, each of which in turn increases the number of objects in orbit, and so on. The problem is particularly exacerbated in low Earth orbit (LEO), where many mega-constellations are planned to be deployed into already crowded orbital regimes.

Traditionally, space situational awareness (SSA) sensor networks have relied on dynamic programming theory to generate tasking plans which govern how sensors are allocated to observe RSOs [3]. However, these techniques do not scale well with many thousands or tens-of-thousands of RSOs. Each new RSO that a network is required to track increases the complexity of the tasking problem. Sensor network capacity must increase by adding sensors, increasing efficiency within existing networks, or a combination of both. With the oncoming metaphorical, and occasionally literal, explosion in the RSO population, improvements in SSA sensor management techniques are necessary.

Over the past decade, breakthroughs in machine learning (ML), a subfield of artificial in-

telligence (AI), have driven successful applications in a wide variety of fields [4]. Machine learning, with its ability to interpret large amounts of high-dimensional data, presents a potential alternative to traditional optimization methods typically used in the SSA sensor management problem.

The remainder of this chapter introduces the SSA sensor allocation problem, provides historical and operational context around SSA tasking algorithms, describes an approach to improving the state-of-the-art using reinforcement learning (RL), a subfield of ML, and outlines the research presented in this document.

1.1 Space Situational Awareness

Space situational awareness is generally defined as the mission of tracking, cataloging, characterizing, and predicting the motion of RSOs [5, 6]. There are many sub-missions within the umbrella of SSA, including maneuver detection, probability of collision assessment, and data association [7]. The work presented here examines the tasking and scheduling submission, specifically the tasking portion, also referred to as the sensor allocation or catalog maintenance problem [6, 7, 8, 9]. The goal of the sensor allocation problem is to minimize uncertainty in the estimated dynamic states (position and velocity) of a catalog of RSOs given a network of sensors.

There are a handful of SSA networks in the world. SSA networks are comprised of mostly terrestrial-based radar and electro optical (EO) sensors, and a few space-based EO sensors. Most networks are operated by governments, but there are some non-governmental organization (NGO) networks, and commercial SSA data providers are also emerging [2, 10, 11]. The United States Space Surveillance Network (SSN) is the leading source of SSA data for the world, and is the most well-documented SSA network [3, 6].

As the number of objects in Earth orbit increases, the demand for SSA data is increasing as well [11]. This has allowed commercial SSA providers to specialize in specific phenomenologies (e.g., radar, EO, radio frequency (RF)) or in making derived products from lower-level data, rather than performing the full data-collection-to-product chain. Universities and governments that had previously not had a significant presence in space are also developing SSA capabilities. Space situational awareness networks are also increasingly linked, with data sharing agreements becoming increasingly more common in the past decade. Many of the upcoming mega-constellations are planned to have maneuverable satellites, requiring more persistent coverage by SSA networks. The demand for persistent, not necessarily exquisite, data is driving sensor operators to expand networks by installing commercial off-the-shelf (COTS) sensors in traditionally under-covered regions, such as the southern hemisphere.

One commonality among all SSA networks is that they all have areas of limited temporal and spatial coverage. The locations of terrestrial sensors are subject to geographic and political constraints. For example, most of the the United States SSN sensors are located in the northern hemisphere, which limits their ability to track RSOs in the southern hemisphere [10]. Electro optical sensors require favorable lighting conditions to observe RSOs, and so generally cannot be used during the day or in the presence of cloud cover. All sensors require occasional maintenance, which can reduce capacity or take them offline for extended periods. Therefore, any SSA network modeling cannot assume omniscience. There exist sets of RSOs that are either not monitored or intermittently monitored. This irregularity in visibility implies that tasking sensors to particular satellites at some times will be better (by some measure) than tasking at other times.

1.2 Tasking Algorithms

Sensor management, also called sensor allocation, is an application of resource allocation theory in which the problem is to determine how to map sensors to objectives to achieve some objective [12]. Sensor management is performed by a distributed network of sensors controlled by a central planner. In SSA, the sensors can be terrestrial or space-based, and are usually radar or EO.

Sensor management can be divided into tasking and scheduling functions [7]. In this dissertation, tasking is the process of the central planner assigning sensors to objectives. In SSA, the objectives can be either specific RSOs or a sensor pointing direction. The SSN tasking algorithm assigns sensors to RSOs. Scheduling is performed at the sensor level, where an individual sensor sequences its tasks over a time period and determines other specifics for the observation, such as integration time for EO sensors.

The SSN operates with centralized tasking and decentralized scheduling. The central planner assigns tasks to sensors, and the sensors schedule those tasks independently. It is not required to decouple tasking and scheduling, but because of the prominence of the SSN and lack of public information about other networks, the approach is the taken in this research. The focus of this research is on tasking.

Operational SSA sensor network tasking algorithms are generally not publicly available, with the partial exception of the SSN. Details of the SSN tasking policy are omitted in published work such that the general structure of the algorithm is described, but the exact performance cannot be reproduced [3]. Therefore, when contextualizing tasking algorithms in research, comparisons are typically made to theoretical benchmarks. A common benchmark tasking policy is to task a sensor to the RSO with the largest state uncertainty at that time, known as the “greedy” policy [9, 13, 14, 15].

The greedy policy, and other benchmarks like it, are myopic in two senses. First, myopic policies only consider the current state of the system. Second, they do not consider other sensors in the network. This myopic property drives inefficiencies in tasking. For example, if two RSOs are in the field of regard (FOR) of a sensor, but one is ascending and the other descending, the myopic policy may choose the ascending RSO even though the descending one will be out-of-view sooner. Another example: if two RSOs are in the FOR of a sensor, and one of them will soon enter the FOR of another sensor, whereas the other will not be accessible for the rest of its orbit, the myopic policy may choose the first RSO, even though the the other one will be inaccessible to the network for longer.

Myopic policies provide an intuitive benchmark with which to compare experimental policies. Furthermore, sophisticated policies, such as those based on dynamic programming, generally have tuning parameters that can affect performance, making benchmark comparisons more complex. Myopic benchmarks are straightforward to reproduce and provide a consistent baseline for comparison.

1.3 Reinforcement Learning

Machine learning is a field that applies statistical methods to learn, from experience, to improve performance in some task [16]. Learning to perform a task can be thought of as estimating a function. Reinforcement learning is one of the three categories of ML, the other two being supervised and unsupervised learning. Where supervised and unsupervised learning seek to generalize predictions from or find structure in a provided data set, respectively, RL estimates a function by interacting with an environment [17]. Reinforcement learning has seen much success in gaming and robotics applications, such as Go, Atari 2600 games, learning to run, and aircraft control [18, 19, 20]. In all of these applications, the number

of possible environment states is huge, which makes solving difficult for traditional optimization techniques. For example, the board game Go has approximately 2.1×10^{170} legal positions [21]. Prior to breakthroughs in RL in 2016, no computational technique had been able to beat professional human performance in Go [19].

In RL, the function to be estimated is the policy, which maps input observations to output actions. There are three components used in any particular RL technique: the policy, the learning algorithm, and the environment. A policy π maps a perceived state s to an action a, which is applied to the environment [17]. With each step of simulation, the environment outputs an observation $\mathcal{O}$ and reward r to the agent, while undergoing a state transition. The learning agent seeks to maximize the cumulative reward by modifying the policy over multiple training iterations. Training generates experiences, which are used to update the policy. During training, RL algorithms balance exploration with exploitation to generate experiences which are useful to optimize the policy. Exploration is purposefully traversing uninvestigated areas of the state/action space to discover areas of greater reward. Exploitation is staying in areas of the state/action space which have the best known reward.

Policies can be represented according to one of two constructs: action-value methods or policy gradient methods [22]. In action-value methods, the policy is a function that maps perceived states to rewards. These methods estimate the value function, which quantifies the potential reward given the state of the environment. In policy gradient methods, the policy is a function that maps actions to rewards. These methods estimate the parameters of a policy, and require the designer to pre-select the structure of the parameterization. In either case, observations may or may not reveal the all the states of the environment, and the policy may or may not have access to a model of the environment. This study uses a particular policy gradient method, proximal policy optimization (PPO), detailed in Ch. 2.

One way to parameterize a function, and in the case of RL, a policy, is with a neural network

(NN). Neural networks are the driving tool behind the recent rise of ML [4]. Machine learning techniques that use NNs as the function estimator are called deep learning (DL) techniques, owning to the common practice of using many layers of neurons in a NN to create a "deep" network. Correspondingly, RL techniques that use NNs as the function estimator are called deep reinforcement learning (DRL) techniques. Invented in the 1980s, DRL was popularized in 2013-2015 with DeepMind's invention of the deep Q learning (DQN) algorithm [23, 24, 25, 26]. Due to the prevalence of NNs across ML, the term "DL" has become synonymous with ML, and DRL is more commonly referred to simply as RL. This document follows that convention, and generally refers to RL to mean DRL. Exceptions to this convention are explicitly called out.

Once a policy parameterization is selected, usually an NN, an algorithm must be selected to optimize the weights of the parameterization to maximize the reward. There are many algorithms that govern how the agent learns about the environment, broadly divided into two categories: tabular and approximate [27]. Tabular methods are generally used for simple environments, and are not the focus of this study. Approximate methods are used when an environment's state and/or action spaces are large, such that exploring a sufficiently representative portion of the space(s) would take a prohibitively long time. Approximate methods do not search for a global optimum, but instead search for a solution that is "good enough". Like any numerical optimization routine, approximate methods can fall into local optima, which is the chief trade-off for the ability to solve large problems.

The final ingredient to an RL technique is the environment. To train a policy, meaning to optimize the weights of an NN, a training algorithm requires data to correlate observations to rewards. Because the training process requires large quantities of data, RL agents are typically trained on a model of the environment, which can simulate many agent-environment interactions inexpensively.

1.4 Suitability to Sensor Allocation

In the sensor allocation problem, the number of possible sensor-target mappings increases on $\mathcal{O}(M^N)$, where M is the number of sensor and N is the number of targets. If time is included, the number of possible mapping trajectories increases on $\mathcal{O}(TM^N)$, where T is the number of time steps in a period. The representation of the sensor-target mapping in the SSA sensor tasking case is $\mathcal{A} = [\mathcal{N}^0, \ldots, N^m \ldots, \mathcal{N}^M]$, where $\mathcal{A}$ is the action space and $\mathcal{N}^m$ is the set of targets in the catalog available to sensor m. It is combinatorially intractable to compare all possible sensor-target mappings. This phenomenon is known as the Curse of Dimensionality [28].

While calculating a tasking solution can be done reasonably quickly with small numbers of sensors and targets, as the population of both (particularly RSOs) increases, the number of potential solutions results in a combinatorial explosion. Reinforcement learning techniques, with their ability to be trained on simulated environments, which are readily available for the SSA sensor allocation problem, and demonstrated performance in other fields, have potential to exceed performance of deterministic methods.

1.5 Research Objectives

The research presented in this dissertation developes techniques for encoding an SSA environment model to apply RL to the sensor allocation problem. The objectives of the research are to develop an environment tailored for the tasking problem, to develop a reward function that balances whole-catalog uncertainty with near-term uncertainty, and to examine the effects of incorporating forecast state knowledge into the observation space.

Tasking-Tailored Environment

The observation and action spaces for an environment model must be designed to interface with an RL agent. The research presented in this paper examines how such interfaces can be tailored to suit the SSA sensor allocation problem, and the impacts of those design constraints and decisions. Much of this objective reduces to the design of the action space and how that informs the design of the observation space, reward function, and training techniques.

Uncertainty-Balanced Reward Function

The reward function is a critical component in any RL scheme. As is described in Ch. 2, the SSA sensor management literature uses reward functions which encourage reductions in either immediate or global catalog uncertainty. The objective of the research in this document is to develop a reward function that balances whole-of-catalog uncertainty with near-term (or local) uncertainty.

Forecast Features in Observation Space

An advantage that the SSA sensor allocation problem has in providing useful information to the RL agent is that orbital dynamics are mostly predictable. Because orbit dynamics are predictable, the position and velocity of RSOs can be forecast into the future with reasonable accuracy. Providing the RL agent with future state knowledge should, intuitively, allow the agent to make farsighted decisions that a myopic agent would otherwise not be able to. An objective of the research in this document is to develop a technique for incorporating forecast states into the features of the observation space.

1.6 Dissertation Organization

The remainder of this document is organized as follows. Chapter 2 details the background information necessary to contextualize the unique research presented in the other chapters. This dissertation is the compilation of two separate but related studies. The first study, called the action masking study, is detailed in Chs. 3 and 4 [29]. The second study, called the environment encoding study, is detailed in Chs. 5 and 6. Chapters 3 and 5 present the methodologies of their respective studies; Chs. 4 and 6 present the results, analysis, and conclusions. Chapter 7 provides an overall perspective on the application of RL to the SSA sensor allocation problem.

Chapter 2

Background

This chapter presents the background information necessary to understand the theoretical foundation and historical context of the unique research presented in the remaining chapters of this dissertation. Section 2.1 discusses the theory of selected topics that are used later in this document. Section 2.2 presents a survey of the recent literature in the topic of RL applications to SSA sensor allocation, as well as the related problem of Earth-observation sensor management. Section 2.3 discusses trends and opportunities for improvement in the literature. Section 2.4 outlines the unique contributions of the new research presented in this dissertation.

2.1 Preliminaries

This section presents selected background topics relevant to the research presented in this dissertation. This section focuses on building a minimum taxonomy and common vocabulary around the topics of SSA, sensor management, and RL. The focus is on theory; application is discussed in Sec. 2.2. While not a comprehensive review of all topics related to the relevant fields, this section provides enough of a foundation so that the reader can understand the research presented in the remaining chapters of this dissertation. Where appropriate, references to further reading are given which provide more detailed information.

2.1.1 Space Object Tracking

SSA is typically performed by radar and EO sensors, although RF and laser range finding are emerging as potentially useful sensor technologies [7, 11]. This document focuses on radar and EO sensors, which are the types of sensors used in the SSN. Radars typically output measurements in azimuth, elevation, range, and range-rate; EO sensors in azimuth and elevation. The measurements are converted into state estimates with a filter, described later.

Estimates derived from radar measurements have less uncertainty in range than do EO-derived estimates because radar measures range directly, whereas EO-derived estimates recover range from multiple azimuth-elevation measurements. Radar is preferred to track RSOs in LEO because of the angular rate requirements. RSOs in low orbits have a higher relative angular velocity relative to terrestrial sensors, and phased array radars are able change beam scan angles much faster than a telescope is able to slew. Electro optical sensors are generally used to track targets in geostationary Earth orbit (GEO) because they have superior angular resolution to radar at the large distances of GEO. Errors for all sensors are typically greatest in the in-track direction (the direction of motion) because the in-track component of velocity is so much larger than the cross-track components, which is amplified by sensor and dynamics noise [7].

Data processing occurs at the sensor level to refine raw measurements (e.g., extracting the centroid of a pixel blob in an EO image) and convert them into a common coordinate frame. The measurements are sent to the filter, usually an extended Kalman filter (EKF) or unscented Kalman filter (UKF) depending on the publication, which combines data from models of the sensors and RSOs to estimate a dynamic states [7]. This document focuses on the UKF. In a Kalman filter, the previous state estimate is propagated forward in time

to calculate a predicted measurement. This predicted measurement is then compared to the actual measurement from the sensor. The predicted and actual measurements are weighted according to their respective expected noise and combined to produce a final state estimate and an associated error covariance. The differences in the various Kalman filters are in how the filter predicts the state estimate and measurement prior to updating with new measurement information. The UKF uses the true non-linear dynamics to predict the a priori state and measurement. The UKF is described in detail in many texts; a couple of excellent references are [30, 31].

There are sources of error at all stages of processing raw measurements into state estimates. At the sensor level, thermal gradients across of the elements in a phased array radar or the mirror of an EO sensor can bias the knowledge of the pointing direction. Atmospheric turbulence can distort the phase of a signal, radar or light, to the sensor, increasing uncertainty in the measurement. In the filter, errors in the dynamical model of the RSO can incorrectly influence the predicted state, increasing the error in the estimate [7].

A method to quantify the uncertainty in the state estimates of RSOs is needed to evaluate sensor tasking algorithms. A thorough review of different uncertainty quantification techniques, including their various use cases, is presented in [7]. This document uses the error covariance matrix, a commonly used technique in the literature, to quantify estimate uncertainty [7, 9, 14, 32]. The covariance matrix P is defined as

$$P = \mathbb{E}[(x - \hat{x})(x - \hat{x})^T] \tag{2.1}$$

where x is the true state, $\hat{x}$ is the estimated state, and $\mathbb{E}$ is the expectation operator [30].

In the RSO tracking problem, the covariance represents a 6-d ellipsoid in the Earth-centered inertial (ECI) frame. The covariance P is a 6×6 matrix, where the upper- and lower-three

diagonal elements represent the positional and velocity components of the uncertainty, respectively. A baked-in assumption of the Kalman filter is that the estimate error is normally distributed. This assumption allows the estimate error to be fully-described by the covariance. In reality, the error is not normally distributed because orbital dynamics, even in the most simple case of Keplerian motion, are nonlinear. However, if uncertainty is relatively small, the covariance is a good approximation that is useful in most cases. For a review of the realism and and trades involved in using the covariance to quantify uncertainty in SSA, see [7].

The research in this document focuses on tasking algorithms, not filter performance or sensor realism. Accordingly, phenomenology specifics are not modelled. The specific phenomenology of sensors is abstracted, and measurements are generated with a selected noise parameter. This approach ensure that the research evaluates differences in tasking algorithm performance, avoiding complications from sensor model fidelity or design.

2.1.2 Dynamic Programming

Dynamic programming is the field of studying multi-stage decision processes [28]. The SSA sensor allocation problem can be described as a multi-stage decision process; each discrete time step is a stage, and the allocating of sensors to targets is a decision or series of decisions. The sensor allocation problem is generally modeled as a Markov decision process (MDP), a class of multi-stage decision processes and which are foundational tools used in RL [33]. Markov decision processes are characterized as memory-less processes, meaning that the state s_k of the system at time k fully describes the system, and all future states can be predicted from state s_k and the corresponding action a_k. A policy is a sequence of transformations $\pi = \{\pi_1, \pi_2, \ldots, \pi_T\}$ that that map the state s_k to an action a_k, mathematically defined as

$$\begin{aligned} a_1 &= \pi_1(s_1) \\ a_2 &= \pi_2(s_2) \\ &\vdots \\ a_T &= \pi_T(s_T) \end{aligned} \tag{2.2}$$

where T is the number of stages (i.e., steps in an episode) [34]. If the functions in a policy are identical $\pi_1 = \pi_2 = \ldots = \pi_T$, as is usually the case, the notation can be shortened to $\pi = \{\pi\}$. The return earned from a T-stage process is

$$R = \sum_{k=1}^{T} R_k(s_k, a_k) \tag{2.3}$$

where R_k is the return earned from a one-stage process. Equation (2.3) can be expanded to separate the return of the final stage from the preceding stages,

$$\begin{aligned} R &= R_T(s_T) + \sum_{k=1}^{T-1} R_k(s_k, a_k) \\ &= R_T(s_T) + R_{T-1}(s_1, a_1) \end{aligned} \tag{2.4}$$

where $T \geq 2$ and the final action a_T has been omitted from $R_T(s_T)$ because the final return is independent of the action. The Bellman equation is a recursion that defines the optimal return R^* for a multistage decision process. The discrete version of the Bellman equation is

$$R^*(s_1) = \max_{\pi}[R_T(s_T) + R_{T-1}(s_{T-1}, a_{T-1})] \tag{2.5}$$

where R_T must be solved recursively to calculate R^* [33, 34]. The optimal return R^* is unique, but the policy π to achieve the value is not. There may be many policies that

can achieve optimal return. The Bellman equation must generally be solved with numerical methods, as most interesting problems are not analytically tractable. Equation (2.5) is the equation that we seek to solve in the SSA sensor allocation problem.

2.1.3 Proximal Policy Optimization

Actor-critic methods are a class of RL training algorithms in which the policy and value function are separate and trained together [22, 35]. The policy is the actor, and the value function is the critic. The policy proposes an action based on observations, and the value function evaluates the quality of the action based on the estimated value of the state. The actor and critic structures can be different, although much of the time they are the same.

PPO is a popular actor-critic technique known for its simplicity and efficacy [35, 36]. PPO can use either continuous or discrete action spaces, and is suitable for multi-dimensional action spaces like the multi-agent SSA sensor allocation problem. PPO operates on the concept of trust regions; it prioritizes small changes in policy to mitigate overshooting effects caused by large step sizes in gradient descent. Given the objective function

$$L(\theta) = \hat{\mathbb{E}}[\log \pi_\theta(a|s)\hat{A}] \tag{2.6}$$

where $\pi_\theta(a|s)$ is probability of a policy with θ parameter vector returning action a given state s, and $\hat{A}$ is function estimator, an RL algorithm seeks to optimize L through stochastic gradient descent (SGD). PPO modifies modifies the objective function in two ways. First, PPO replaces the policy probability with the probability ratio $r(\theta)$, which represents the difference between two policies:

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta,\text{old}}(a|s)} \tag{2.7}$$

where $\pi_{\theta,\text{old}}$ is a reference policy. Second, the loss function is clipped such that moving $r(\theta)$ away from one is penalized:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}[\min(r(\theta)\hat{A}, \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A})] \tag{2.8}$$

where ϵ is a small positive number and the clip function is defined as

$$\text{clip}(x, a, b) = \begin{cases} a, & x < a \\ b, & x > b \\ x, & \text{otherwise} \end{cases} \tag{2.9}$$

where $a < b$. The crucial term in Eq. (2.8) is $\min(r(\theta)\hat{A}, \text{clip}(\cdot)\hat{A})$, which takes the lesser of either the the base loss value $r(\theta)\hat{A}$ or the clipped value. This term makes the estimate update pessimistic; it can make only small changes to drive toward improvements, but it can make large changes to drive away from extreme loss [36].

Each training iteration, some fixed number of experience tuples (observation, action, reward) are collected, then stochastically sampled to optimize the critic. The actor is updated by copying the weight from the critic, optimizing the policy's performance.

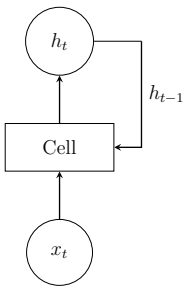


Figure 2.1: Recurrent neural network cell (adapted from [37]).

2.1.4 Long Short-Term Memory

There are many types of NN structures that are used in ML. Recurrent neural networks (RNNs) are one type that is useful for ingesting sequential data, such as time series or text [37]. Recurrent neural network cells, which collectively make up the layers of a network, have a feedback loop which allows the cell to maintain memory from previous inputs. A cell accepts an input from a previous layer, with a simultaneous input from the cell's own "hidden" state, which is a function of previous inputs. The hidden state, which is updated along with the regular weights, serves as the memory of the cell. An RNN cell is illustrated in Fig. 2.1

One of the drawbacks of "vanilla" RNNs, and indeed with all NNs, is that as successive data are ingested, the network is susceptible to vanishing or exploding gradients. Vanishing or exploding gradients can cause stagnation or instability, respectively, in training.

Vanishing gradients are caused when the inputs to a NN are less than one; successive multiplications cause the gradient to approach zero. The long short-term memory (LSTM) cell (and corresponding LSTM network) is a type of RNN that is designed to combat the vanishing gradient problem [38]. Long short-term memory cells have two internal states, a hidden state (the same as the basic RNN) and a cell state. The cell state is used in combination

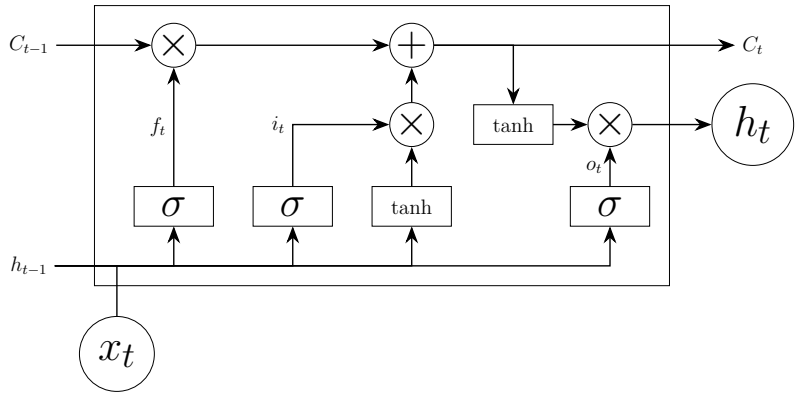


Figure 2.2: Long short-term memory cell (adapted from [39]).

with a set of gates to to control what sequences of the data the cell should "forget" and "remember." This architecture allows the LSTM cell to associate inputs with relevant sequences seen long before the current input (long term memory). The net effect is that an LSTM network can output data relevant to the current input sequence, even if the data relevant to that sequence has not been seen recently. An LSTM cell is illustrated in Fig. 2.2.

The equations governing the LSTM cell are

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.10a}$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.10b}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.10c}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tanh(W_C[h_{t-1}, x_t] + b_C) \tag{2.10d}$$

$$h_t = o_t \odot \tanh(C_t) \tag{2.10e}$$

where x_t is the input, h_{t-1} is the previous hidden state, σ is the sigmoid function, W is a weighting matrix, b is a bias vector, $\odot$ is the Hadamard product (element-wise multiplication), and the outputs of the equations are explained next. The forget gate f_t determines

which information should be removed from the cell state C_t. The input gate i_t determines which information from the input x_t should be included in the cell state C_t. The output gate o_t determines which information from the input x_t is relevant to the hidden state h_t. The cell state C_t is updated by combining the information from the forget gate f_t, the input gate i_t, the input x_t, and the previous hidden state h_{t-1} of the cell. The output of an LSTM cell, like a vanilla RNN cell, is the hidden state h_t. The difference is that, in the LSTM, the hidden state h_t is a combination of the output gate o_t and the cell state C_t.

2.2 Literature Review

The field of SSA has a large body of research, but only a relatively small fraction of the research to date is concerned with the sensor tasking problem, and a yet smaller fraction of that subset of work investigates RL techniques. This section provides an overview of topics related the application of RL to SSA sensor management. First, an overview of RL applications to Earth-observation satellite tasking and scheduling, a problem similar to tracking RSOs, is provided. Second, a comprehensive review is given of recent literature that examines the application of RL to the SSA sensor allocation problem. The scope of of both reviews is limited to applications of DRL.

2.2.1 Earth-Observation Satellite Tasking and Scheduling

Earth-observation satellite tasking and scheduling is a well-studied problem in which a satellite must select from and order a set of terrestrial targets to observe within a given time frame. Much research has been done in applying RL to Earth-observation satellite tasking and scheduling [40, 41, 42, 43, 44, 45, 46, 47]. Earth-observation satellite tasking and

scheduling is similar to SSA sensor management in that both are resource allocation problems, have visibility constraints driven by orbital dynamics, and can be partitioned into tasking and scheduling sub-problems. The chief difference between the two problems is that in Earth-observation it is assumed that the target dynamic states are known, so the factors of target state estimation and uncertainty are not present as they are in the SSA problem. However, the similarities between the two problems are still useful to examine, in particular how the environment is encoded to interact with the RL agent. The encoding defines the observation and action spaces of the environment, and the reward function. In this section, the Earth-observation problem is assumed to be single agent; only one satellite is used. This section is focuses on the research that applies *deep* RL to the Earth-observation tasking and scheduling problem.

2.2.1.1 Action Space Encoding

The objective Earth-observation tasking/scheduling problem is twofold: to select targets from a catalog that is too large to observe all in the given time period, and to schedule observations of the selected targets. These two sub-problems are the same as in SSA sensor management, except the targets are in space. There are many ways to structure the problem to apply RL. Scheduling can be abstracted, focusing the RL techniques on the tasking sub-problem [42, 44, 45]. The tasking and scheduling sub-problems can be solved sequentially, with a dedicated NN for each function [43, 46, 47]. In [40, 41], tasking and scheduling are combined into a single problem. Each of these approaches has different implications on the way in which the environment is encoded to interact with an RL agent.

Tasking is a discrete process in which actions are selected from a set of options. In most cases, the action is a location on the Earth's surface, either defined by a latitude and longitude [40, 42, 46, 47] or by a grid cell [44]. Other representations are possible; for example, [45]

uses a grid of antenna pointing directions relative to the satellite body frame as the action space. Actions can be selected by an RL agent one-at-a-time, as in [40, 42, 43, 44, 45]. Alternatively, actions can be made in batch form, where a list of targets are associated with some time window, as in [46, 47]. In the terminology of RL, all of these representations are categorized as discrete action spaces. In [42, 43], the action space is further reduced to a binary case, where the agent must select whether to accept or reject a new tasking request given a pre-existing tasking plan. The discrete action space for the SSA sensor management problem is to to represent each RSOs as an action. The same single, batch, and reduction methods may be applied to the SSA scenario.

Scheduling in the Earth-observation problem is the process of determining a time at which to start an observation period. The problem is identical in SSA sensor scheduling. This process is continuous, as time is a continuous variable. The duration of the observation period is a constraint of the problem, and so the start of the observation time is the only variable. The scheduling process is often abstracted to focus on the tasking sub-problem. When this is done, the problem is constructed with discrete time steps and synchronized actions, as in [40, 42, 44, 45]. In these cases, actions are always taken on the time step, meaning the fidelity of the schedule is limited by the duration of the time step. This approach intrinsically solves the scheduling sub-problem without having to formulate a separate optimization scheme. In [42], another approach is taken, where the observation window is heuristically assigned to a future continuous time after a tasking decision is made. In either case, these approaches do not explicitly optimize the time of observation, but rather focus on allocating observation opportunities among targets.

When the schedule is explicitly optimized, the RL agent searches for a continuously-valued time to start an observation period which maximizes some utility function. This technique is used in [41, 43, 46, 47]. The action space in these cases can include multiple variables if the

tasking solution (a discrete variable) is also part of the optimization. For example, in [46] the tasking agent selects a batch of targets to observe, then the scheduling agent determines observation window start times for the selected targets, and may optionally remove some targets from the list. The action is a tuple of the filtered list of targets and the times at which those targets' observation windows will start.

The discussed approaches in defining the action space for the Earth-observation problem are all applicable to the SSA sensor management problem. The action space can be discrete, in which each RSOs corresponds to an action, or continuous, in which the action is the start time of the observation. The action space can also be a discretized version of a continuous space. For example, similar to [45], the FOR of an SSA sensor can be discretized into a grid of azimuth-elevation directions, where the agent must select one per time step (this is further examined in Sec. 2.2.2). The action space can also be hybrid, by combining observation start time (continuous) with an RSO (discrete). In any case, the choice of action space is informed by the objective of the design, whether that be tasking, which favors a discrete approach; scheduling, which favors continuous spaces; or some combination of the two.

2.2.1.2 Observation Space Encoding

Relative to the wide variety of action space encodings, the representation of the observation space is fairly consistent across the examined literature [40, 41, 42, 43, 44, 45, 46, 47]. The observation space of the Earth-observation scheduling/tasking problem is generally represented as a set of tuples, where the number of tuples is the number of requested tasks (locations on the Earth), and the elements of the tuples are attributes such as availability window timing, expected reward, and resource requirements (e.g., power, data storage). This method of encoding lends itself to the SSA sensor allocation problem, where the tasks are also discrete targets that have availability windows, expected reward, and possibly resource requirements,

depending on the design and fidelity of the environment model. In the Earth-observation problem, the expected reward for imaging a target is generally assumed known a priory, and is represented by a priority value. In the SSA sensor allocation problem, the expected reward is less-well defined, or at least there is less consensus on how to define it, which is discussed more in Sec. 2.2.2.

The observation space in the Earth-observation problem may include the geodetic coordinates of the target, which provides spatial information to the RL agent [40, 44, 46]. Spatial information is useful in the feature space (a general term for observation space) of ML problems because it provides context to the learning agent. For example, in text-recognition, words that are far away from each other in a sentence are less likely to be related than words that are close together. In the Earth-observation problem, targets that are near each other can be imaged close in time because the required slew time is small. In [44] the spatial framing of the problem is exploited using a convolutional neural network (CNN), which are specifically designed for spatial data. Spatial information is not as readily available in the SSA sensor allocation problem. Because RSOs are always moving and usually in different orbits, the spatial relationships between targets are dynamic (the notable exception being GEO constellations). Spatial relationships are further complicated if multiple or space-based sensors are in play. Coordinate transforms can be used to represent spatial data more directly, such as representing the RSO state in the sensor FOR frame, which is discussed in Sec. 2.2.2.

In the Earth-observation tasking and scheduling problem, the availability windows between sensor and targets are known ahead of time, and is represented in the observation space of all the reviewed literature [40, 41, 42, 43, 44, 45, 46, 47]. Availability windows, also called "access windows," are a commonality between the Earth-observation and SSA sensor allocation problems. In both cases, availability windows are known ahead of time. However,

as is discussed in Sec. 2.2.2, none of the existing SSA literature includes availability windows in the observation space. In the Earth-observation problem, the dynamic states of both the sensor and targets are assumed known. In the SSA case there is uncertainty on the window timing because the state of the targets is not perfectly known (the state of the sensor is assumed known). This discrepancy between the true state and the estimated state in the SSA problem is a key difference between it and the Earth-observation problem. Because estimated states are updated throughout an episode, the availability windows are dynamic, and so using availability windows as a feature in the observation space must be adapted for use in the SSA sensor allocation problem. The concept of including forecast availability windows is one of the contributions of the research presented here.

2.2.1.3 Summary

There are infinitely many ways to represent the environment of a tasking and scheduling problem to an RL agent. The representation is driven by the objective of the experiment. In this research, the tasking sub-problem is the focus, and the environment is encoded to reflect that by using a discrete action space. Because SSA networks use multiple sensors, unlike the reviewed Earth-observation literature, this research uses multiple discrete action spaces linked by a single agent. This research borrows the concept from the Earth-observation problem of using forecast availability windows to inform the agent. The details of the environment representation for this research are described in later sections.

2.2.2 Space Situational Awareness Sensor Management

This section reviews the current state of research in applying RL to the SSA sensor allocation problem. Papers are presented in roughly chronological order and by author lineage, and

are grouped by similarity where applicable. Each review highlights the following aspects of the paper: the scenario parameters (e.g., number of RSOs, orbit regime, sensor FOR), the reward function, the learning algorithm, the NN architecture, the observation and action spaces, the efficacy of the trained policies, and which benchmark policies, if any, they were evaluated against. This section is limited to research that applies *deep* RL to the SSA sensor allocation problem.

In [32, 48] actor-critic (AC) and asynchronous advantage actor-critic (A3C) are used to track 3 to 300 RSOs in GEO from a single, omniscient sensor; the sensor has access to every RSOs at all times. Dynamics are propagated via 2-body motion. All satellites and the sensor are restricted to planar motion. An EKF is used to estimate dynamic states of RSOs. The agent receives a penalty (negative reward) every step in which any RSO's trace of positional covariance is above a threshold, and a reward when the entire catalog is below said threshold. The reward function for [48] is

$$r = \begin{cases} \alpha, & \max\limits_{N} \operatorname{tr}(P^n_{1:3}) \leq \varepsilon \\ -\beta, & \min\limits_{N} \operatorname{tr}(P^n_{1:3}) > \varepsilon \end{cases} \tag{2.11}$$

where $P^n_{1:3}$ is positional covariance of target n, N is the number of RSOs, and α, β, and ε are positive scalars. In [48], $\alpha = \beta = 1$.

The NNs for the actor and critic have the same hidden layers structure: 3 fully-connected (FC) layers of $[200, 100, 50]$ nodes. The activation function is rectified linear unit (ReLU). The observation space consists of the estimated ECI state and the diagonal elements of the covariance matrices of all RSOs. The action space of the agent is choose one of N RSOs to task. The trained network generally maintains all RSOs' covariance below the threshold; no comparisons to a baseline policy are made.

In [14, 15], the depth and scope of [32, 48] are significantly expanded. Both [14] and [15] are summarized here as a single entry due to their similarity. The dynamic propagation model is upgraded to Simplified General Perturbations 4 (SGP4), the filter is a UKF, and the number of RSOs is increased to 100 to 400. Low Earth orbit populations are studied in addition to GEO. The single light detection and ranging (LIDAR) sensor has a limited field of view (FOV), FOR, and slew rate. The sensor receives noisy measurements of azimuth, elevation, and range. The agent is rewarded with the maximum reduction in covariance trace of the RSOs in the grid unit to which the sensor is tasked. The reward function has two key differences from [32, 48]: the reward is a continuous (vice binary) value and the reward is a function of the set of visible RSOs (vice the whole catalog). The reward function for [14, 15] is

$$r = \max_{\mathcal{N}_{\text{vis}}^m} [\text{tr}(P_n^-) - \text{tr}(P_n)] \tag{2.12}$$

where $\mathcal{N}_{\text{vis}}^m$ is the set of RSOs in the FOV of sensor m at a single time step, and P_n^- and P_n are the a priori and a posteriori covariance, respectively, of target n. Equation (2.12) is a surrogate for information gain, which is a measure of reduction in uncertainty.

Notably, the observation space of the agent has more information than the historically commonly-used dynamic states and covariance matrices of the RSO catalog. The observation space is a $90 \times 19 \times 11$ grid, with "information layers" containing features with data about all the RSOs in the given grid unit. The features include metrics such as number of RSOs in a grid and rate of change in azimuth of each RSO in a grid. The action space is a 90×19 discretized grid of the sky representing 1710 possible actions.

Proximal policy optimization is used as the training algorithm and six NN architectures are studied. The NN architectures vary number, size, and type of layers (FC and CNN layers

are used). The observation space being a grid, which embeds spatial information, allows the use of CNN layers, which had not been used in prior studies. Population-based training is used, in which multiple agents are trained concurrently; the agents are regularly ranked against each other and the highest-performing agents' parameters are copied to the lower-performing agents. Notably, the training is implemented using Ray, a standard library built for distributed computing with ML [49]. The various NN-based policies are compared to two myopic policies: "greedy" tasks the maximum trace of covariance matrices, and "advanced greedy" incorporates the slew time of the sensor into the tasking decision. All RL policies outperform the both myopic policies in terms of mean trace of covariance and unique RSOs observed.

In [13, 50], the techniques developed in [14, 15] are applied to multiple orbital sensors. Both [13] and [50] are summarized here as a single entry due to their similarity. Only the components different from [14, 15] are highlighted here. These are the only of the reviewed studies to use multiple sensors, and the only to use space-based sensors. An agent is trained on an environment with a single LEO sensor, using identical training parameters as [14, 15]. The sensor has an Earth-exclusion cone and pointing directions are discretized into a 90×26 grid. Two reward functions are examined, Eq. (2.12) and a modified version that disincentives frequent sensor slews.

The environment is adapted to a multi-agent framework by duplicating the trained agent across three LEO sensors, then evaluating the independent agents' overall performance in single and Monte Carlo simulations. The agents are independently controlled; this is known as a decentralized multi-agent scenario. Agent robustness is examined by evaluating agents trained on one orbital configuration in another. The actions of sensors are not synchronized, so at each actionable time step only one sensor is tasked. This is a more realistic method of propagation. The RL policies generally outperform the myopic policies, with one NN

architecture being a definitive top performer [13, 50].

In [51], the techniques developed in [13, 50] are further advanced by allowing neighboring sensors to share information. Only the new components of [51] are highlighted here. Multiple ground-based sensors are trained on a catalog size of 100 RSOs in geosynchronous Earth orbit (GSO). Each sensors' observation space, the FOR discretized into a grid by azimuth and elevation, is expanded to include the neighboring East and West sensor' grids, tripling the size of the observation space of a single sensor. The observation space maintains the same "information layers" as in [14, 15], but now extended across 3 sets of grids. This expansion of the observation space does not affect the action space; it is still a single sensor's azimuth-elevation grid. This structure is decentralized, but allows the agents to share data. So while the agents still operate independently, their observations and actions influence each other.

A more sophisticated training algorithm is used, and additional NN structures are examined. Agents are trained in environments with 6 sensors and 100 RSOs. They are then simulated with increasing amounts of sensor and targets, up to 10 and 1000, respectively. The trained policy, despite being trained on 100 RSOs, outperforms benchmark myopic policies on catalog sizes up to around 600, showing good generalization.

In [52] an agent is trained via advantage actor-critic (A2C) on a scenario with 10-40 RSOs. The RSOs are distributed across LEO, medium Earth orbit (MEO), and GEO regimes. Dynamics are modelled via the Joint Earth Gravity Model 3 (JGM-3), including Sun and Moon perturbations. The filter is a linear Kalman filter (KF). A single omniscient sensor is used with a linear Kalman filter as the dynamic state estimator. The reward function is the same as in [32, 48], see Eq. (2.11), except with $\alpha \neq \beta$. The NN structure is similar to that of [32, 48], the only difference being that the size of each layer is decided based on the number of RSOs in the scenario. The observation and action spaces are identical to

[32, 48]. It consists of the estimated ECI state and the diagonal elements of the covariance matrices of all RSOs. The action space of the agent is to choose one of N RSOs to task. The trained NN is compared against a random policy, a myopic maximum Fisher Information Gain (FIG) policy, and a myopic maximum Shannon Information Gain (SIG) policy. The trained policy outperforms all other policies; however, the random policy outperforms both information-based policies, which calls into question any conclusions about the efficacy of the RL-based policy.

In [53] the implementation of [52] is reformulated to ease computational requirements and increase accessibility via the use of open source libraries. Only the major differences from [52] are discussed here. A population of 50 RSOs is used. The paper replaces the linear Kalman filter from [52] with a UKF. The agent receives a negative reward equal to the maximum covariance trace among all RSOs at every step. The reward function is

$$r = -\max_{\mathcal{N}}[\mathrm{tr}(P_n)] \tag{2.13}$$

where $\mathcal{N}$ is the set of all RSOs in the catalog, and P_n is the estimate covariance of the nth RSO.

The trained RL policy is compared to random and greedy policies, where the greedy policy tasks the maximum trace of covariance matrices. The RL policy outperforms the random policy, but not the greedy policy, in terms of the reward function. No other metrics are used to compare the policies.

In [54] a double deep Q network (DDQN) is applied to an environment with 25 LEO RSOs. Dynamics are propagated via 2-body motion. The scenario uses a single sensor which receives noisy angle and range measurements. The filter used is an EKF. The single sensor has limited FOV, FOR, and slew rate. The agent receives a reward for each RSO in the FOV during a

step. The reward function is

$$r = \sum_{n=1}^{|\mathcal{N}_{\text{vis}}|} 1 \tag{2.14}$$

where $|\mathcal{N}_{\text{vis}}|$ is the number of RSOs in the FOV of the sensor at a single time step.

The observation space consists of the difference in azimuth between the sensor boresight and the relative position vectors of the RSOs d_{az}, and likewise for elevation d_{el}. The action space is "choose one of five": move up, down, left, right, or do nothing, referring to the sensor's pointing direction. The paper does not describe the NN architecture used in the study. The trained RL policy is compared to a random policy. The RL policy outperforms the random policy in terms of covariance trace at the end of an episode.

2.3 Trends and Opportunities

This section analyzes the trends and opportunities to improve in the state of research in applying DRL to the SSA sensor allocation problem. It primarily focuses on the research first discussed in the SSA literature review, Sec. 2.2.2, and is informed by the context of the Earth-observation research presented in Sec. 2.2.1. The relevant attributes of the papers discussed in Sec. 2.2.2 are summarized in Tables 2.1 to 2.3. Table 2.1 summarizes the scenarios used in the studies. Table 2.2 summarizes the environment encoding and RL schema used in the studies. Table 2.3 summarizes the reward functions used in the studies. The unique research presented in this dissertation is included in Tables 2.1 to 2.3 for context; the discussion of those entries is in Sec. 2.4.

The trends and opportunities to improve in the research are analyzed in cross-cutting themes: tasking and scheduling, action space, observation space, reward function, training algorithm,

Table 2.1: Summary of scenario configurations in SSA literature review.

Paper	Multiple Sensors	Num. RSOs	Limited FOR	Propagator	Filter	Orbit Regime(s)
[32, 48]	No	3–300	No	2-body	EKF	GEO
[14, 15]	No	100, 400	Yes	SGP4	UKF	LEO, GEO
[13, 50]	Yes	100, 400	Yes	SGP4	UKF	GEO
[51]	Yes	100–800	Yes	SGP4	UKF	GSO
[52]	No	10–40	No	JGM-3	linear KF	LEO, MEO, GEO
[53]	No	50	No	JGM-3	UKF	LEO, MEO, GEO
[54]	No	25	Yes	2-body	EKF	LEO
Chs. 3–4, [29]	Yes	100	Yes	2-body	UKF	LEO
Chs. 5–6	No	2–40	Yes	2-body	UKF	LEO (co-located)

Table 2.2: Summary of RL schema in SSA literature review (x: dynamic state; P: covariance; d_{az} and d_{el}: azimuth and elevation differences, respectively, between sensor boresight and RSO).

Paper	Algorithm	NN Features	Observation Space	Action Space
[32, 48]	AC, A3C	FC	x, P	${}_NC_1$ RSOs
[14, 15]	PPO	FC, CNN	Info. layers	${}_DC_1$ grids
[13, 50]	PPO	FC, CNN	Info. layers	${}_DC_1$ grids
[51]	PPO	FC, CNN	Shared info. layers	${}_DC_1$ grids
[52]	A2C	FC	x, P	${}_NC_1$ RSOs
[53]	A2C	FC	x, P	${}_NC_1$ RSOs
[54]	DDQN	Unspecified	d_{az}, d_{el}	${}_5C_1$ directions
Chs. 3–4, [29]	PPO	FC	diag(P)	${}_NC_M$ RSOs
Chs. 5–6	PPO	FC, LSTM	Many features	${}_NC_1$ RSOs

Table 2.3: Summary of reward function attributes in SSA literature review.

Paper	Equation	Data Type	Description
[32, 48]	Eq. (2.11)	Binary	Keep max catalog uncertainty below threshold
[14, 15]	Eq. (2.12)	Continuous	Maximize information gain
[13, 50]	Eq. (2.12)	Continuous	Maximize information gain
[51]	Eq. (2.12)	Continuous	Maximize information gain
[52]	Eq. (2.11)	Binary	Keep max catalog uncertainty below threshold
[53]	Eq. (2.13)	Continuous	Prioritize largest covariance
[54]	Eq. (2.14)	Discrete	Maximize number of RSOs in FOV
Chs. 3–4, [29]	Eq. (3.3)	Continuous	Minimize mean catalog uncertainty
Chs. 5–6	Eq. (5.24)	Continuous	Multiple components

NN architecture, catalog size, dynamics propagator, and filter.

2.3.1 Tasking and Scheduling

Of the available research that models multiple sensors [13, 50, 51], all merge the tasking and scheduling sub-problems, unlike many of the approaches in the Earth-observation problem (see Sec. 2.2.1). This approach could be because those papers focus on decentralized control, which inherently simplifies the tasking problem by removing the choice of multiple sensors. In any case, the examined papers all approach the SSA sensor management problem from a scheduling-only perspective. The tasking sub-problem is not applicable in the examined papers. This leaves a gap in the research for applications such as the SSN, which has a centralized tasking agent and decentralized schedulers [7].

2.3.2 Action Space

The action space is inherently coupled with the number of sensor and how FOR restrictions are handled. The examined research can be segmented into three groups with regards to

sensor numbers and FOR restrictions: a single sensor with no FOR restrictions, a single sensor with FOR restrictions, and multiple sensors with FOR restrictions. The papers [32, 48, 52, 53] use a single omniscient sensor (see Table 2.1), in which the sensor has access to every RSO at all times (an unlimited FOR). This simplification is less realistic, and is not present in recent publications. The omniscient sensor formulation lends itself to an action space in which the sensor can choose any one of RSO in the catalog (see Table 2.2). This representation would be useful in a tasking framework, where the centralized tasker could map sensors to targets; however, none of the examined papers use such a framework. With an omniscient sensor, there are no invalid actions, which removes a complication from a higher fidelity model.

The group of papers [14, 15, 54] take into account limited sensor FOR, and considers only a single sensor (see Table 2.1). This higher-fidelity technique is an improvement over the omniscient sensor in terms of realism. This technique requires the consideration of invalid actions, actions which are not physically possible, such as tasking a sensor to an RSO on the other side of the Earth. In [14, 15], the discretized grid is limited to the sensor's FOR, inherently preventing invalid action selection. In [54], the invalid actions, such as pointing the telescope below the horizon are overridden with a "do nothing" command. This approach likely hinders training because the agent's chosen action is modified after the fact, briefly breaking the action-state-transition relationship.

The most sophisticated group of papers [13, 50, 51] use multiple sensors with limited FOR, more realistically modelling a real SSA network (see Table 2.1). The sensors are decentralized, which is not realistic, as real SSA networks have a centrally controlled tasking authority [7]. However, decentralized control is its own research area, and can improve resiliency of sensor networks, so the design is still relevant. None of the examined papers use a centralized multi-sensor framework.

2.3.3 Observation Space

Of the papers examined, the less recent ones [32, 48, 52, 53] use the dynamic states and covariance matrices of the RSOs as the observation space (see Table 2.2). Dynamic states are not well-suited as observation features for an NN because they do not provide useful information to the agent. The ECI position and velocity of an RSO does not inherently inform the agent about the uncertainty in the state estimate, nor does it provide sensor-target access information. Observation spaces with dynamic states are relying on the NN to learn the transform to useful information, which is not ideal given that other representations are analytically and readily available. The papers [13, 14, 50, 51], which are more recent, use more directly useful features in the observation space (e.g., azimuth and elevation velocity relative to sensor). This more direct approach allows the NN to focus on optimizing the scheduling problem, rather than transforming coordinates.

2.3.4 Reward Function

The reward functions used in available research are based on covariance, with the exception of [54] which seeks to maximize the number of targets in the sensor FOV (see Table 2.3). The method in [54] treats all RSOs identically; the more targets in-frame, the greater the reward, regardless of covariance. This is a simple reward function, but not well-suited to the sensor allocation problem, which seeks to minimize uncertainty. Maximizing the number of targets in-frame is better-suited for an SSA application in which the goal is to monitor regions of space for activity, rather than observe specific RSOs.

In [32, 48, 52], reward is given only when the entire catalog is below a covariance threshold (see Eq. (2.11)). This approach encourages balancing the uncertainty among the whole catalog, and not over-tasking targets with already low uncertainty. It also theoretically

encourages long-term decision making, as the reward is responsive to cumulative decisions made over the course of an episode. However, because the reward is binary, it leads to a static reward signal. When the catalog is generally well- or poorly-tracked, the signal does not change. This offers an opportunity for improvement in training because actions are not associated with changes in reward.

In [53] the reward is inversely proportional to the maximum trace of covariance in the catalog (see Eq. (2.13)). This approach encourages tasking targets with high uncertainty. However, this technique is highly sensitive to initial covariance values and numerical stability of the filter, both of which could cause very large values in the covariance matrix. In these events, the reward signal would be biased low, and regardless of how well the other targets are being tracked, the agent would be punished for its performance.

In [13, 14, 15, 50, 51], the reward function is based on a pseudo-information gain metric, the reduction in trace of covariance (see Eq. (2.12)). This approach encourages tasking targets with high uncertainty. Because it takes into account only the RSOs visible to the sensor(s), it not susceptible to outliers in the way that [53] is. The information gain approach offers a highly dynamic reward signal, which is conducive for training, as it allows the agent to experience a wide dynamic range of action-reward associations. However, a reward function based only on the states of visible targets is myopic in time. A large information gain in the moment may sacrifice long-term performance if some targets do not have many access windows. Depending on scenario parameters (e.g., catalog size, episode duration) this downside may not be significant.

2.3.5 Training Algorithm

A variety of training algorithms are used in the current research (see Table 2.2), but little discussion is made as to the applicability of chosen technique to the scenario. The most recently-invented technique, and currently popular in the ML field, used in the reviewed papers is PPO [13, 14, 15, 50, 51]. Notably, all of the papers except [54] use policy gradient techniques. Presumably, this is because value gradient techniques do not scale well with the SSA sensor allocation problem.

The papers that demonstrate the most success in training [13, 14, 50, 51] incorporate another layer of sophistication on the training algorithm, namely population based training. This suggests that the SSA sensor allocation problem could benefit from non-standard training techniques. This could be because in the SSA sensor management problem, regardless of action space representation, there are a large number of available actions, most of which are either invalid (i.e., observing a target over the horizon) or provide no benefit (i.e., selecting an azimuth/elevation grid with no targets in it). Many classic RL techniques have difficulty with such action spaces, and so more specialized techniques, such as population-based training, may be necessary, depending on the exact scenario.

2.3.6 Neural Network Architecture

None of the examined papers detail how the NN architectures were chosen[1]. The papers [32, 48, 52, 53] use single configurations of FC networks. The papers [13, 14, 50, 51] use

[1]In [52] there is some discussion of how the number of nodes per layer was chosen, but more in a rule-of-thumb sense rather than any justification as to why the approach was chosen.

multiple configurations of a more sophisticated NN that incorporates CNN layers as well as FC layers. The use of CNN layers lends itself to the spatial nature of the observation space in these papers. These papers have the most rigorous analysis of NN architecture among the examined literature. Notably, none of the current research uses RNNs, which are used for time-series observation data. This represents an opportunity to improve in the research, as the SSA sensor allocation problem, with its well-defined dynamics, could benefit from time series analysis.

2.3.7 Catalog Size

The current research simulates a variety of catalog sizes (see Table 2.1), with the more recent research using larger catalogs [13, 14, 50, 51]. The catalog sizes in the research are low relative to the entire real life RSO catalog, which numbers in the tens-of-thousands, but when compared to specific orbital regimes, the examined catalog sizes are closer to realistic [55]. Furthermore, given the recency and novelty of applying RL, specifically *deep* RL, to the sensor allocation problem, the size of the catalogs examined is less relevant than the other aspects of the problem. The studies that have used the largest number of RSOs [13, 14, 15, 50] make use of Ray, a relatively new ML tool that is in active development, and use super-computing resources[49].

2.3.8 Dynamics Propagator

The current research uses various levels of fidelity of dynamics realism (see Table 2.1). None of the current research examines the effects of dynamics on training. Given that higher-order dynamic effects, such as J2, do not majorly impact motion over short time spans, and that the current research simulates relatively short time spans, the choice of dynamic propagator

is not likely a major influence on the training process or results. One area of research that could be expanded on is examining the generalizability of RL policies trained on low-fidelity dynamics to high-fidelity environments.

2.3.9 Filter

All but one of the reviewed papers [52] uses either an EKF or UKF as the dynamic state estimator (see Table 2.1). The choice of filter, provided it has at least a basic level of functionality, is not a critical design parameter in the current research. The RL policies are measured against themselves and simple benchmark policies, so as long as all policies are using the same filter, policy comparisons are valid.

2.3.10 Summary

In summary, the current research on applying DRL to the SSA sensor allocation problem is relatively nascent. All of the recent research merges the tasking and scheduling sub-problems into a single scheduling problem. This approach could be tailored to be more directly applicable to the centralized-tasking/decentralized-scheduling scheme of the SSN by separating the tasking and scheduling sub-problems. The most sophisticated research uses decentralized, limited FOR sensors that share information with their nearest neighbors [51]. There is no current research that examines the centrally-controlled multi-sensor problem.

The action space representation is coupled with research objectives and the fidelity of the environment. Action spaces are generally one of two types: choose 1 of D pointing directions, or choose 1 of N targets,. The former is best suited to a scheduling framework, the later to a tasking framework, although tasking is not specifically addressed in the examined research.

The trend in observation space features has been to exclude dynamic states, and include more directly useful features such as covariance and relative position information. Observation spaces which discretize pointing directions embed spatial information, which is useful for the RL agent. This type of observation space enables the use of CNNs, and has proven successful in a number of papers [13, 14, 50, 51].

There are a variety of reward function designs in the current research, all with strengths and weaknesses. The reward functions which track some metric of the whole catalog are designed to encourage strategic actions, but are susceptible to outliers. The reward functions which track only the visible RSOs encourage the best immediate action, but must rely on training to embed the long term effects of decisions.

The most-commonly used training algorithm in the research is PPO. The most sophisticated research layers population based training on top of PPO. Neural network architecture analysis is not a strong focus in most of the examined research. The current research uses either simple FC networks or more complex CNN networks (that also have FC layers). None of the current research uses RNNs. Given the well-defined orbital dynamics of the problem and the successful use of RNNs in the Earth-observation research (see Sec. 2.2.1), the lack of RNN use in the SSA sensor allocation problem is an opportunity to improve the research.

Scenario parameters such as catalog size, dynamics propagator, and filter choice are less significant design parameters in the current research.

2.4 Contributions

The unique research presented in this work takes influence from the literature reviewed in Sec. 2.2 and merges some of the ideas to address opportunities for improvement discussed

in Sec. 2.3. This section presents the contributions of the research in this dissertation, highlighting similarities and differences with the literature.

Centralized Tasking of Multiple Sensors

This research is the first to apply DRL to the tasking portion of the multi-sensor SSA sensor allocation problem. Previous work has merged the tasking and scheduling sub-problems, and has done so in a decentralized architecture in which each sensor decides actions on its own [13, 50, 51]. This research uses a single NN as the tasking engine, which sends sends tasks to multiple sensors. This portion of the unique research is presented in Chs. 3 and 4 [29].

Observation and Action Space Formulation for Centralized Tasking

This research developes a unique method for representing the observation and action spaces of the SSA sensor allocation environment, tailored for the tasking problem. The method merges the action space of previous works, which is to choose one of N targets, with an action masking technique used in more general RL applications [32, 48, 52, 53]. This is the first instance reported in the literature of action masking being used to prevent tasking invalid targets in the SSA sensor allocation problem. The method is described in Chs. 3 and 4, then further developed in Chs. 5 and 6 [29].

Forecast Access Windows as Observation Features

Tasking and scheduling for Earth-observation and SSA sensor allocation have many similarities, among them the need to know access windows, and that these windows are driven

by orbital dynamics. Earth-observation research has used forecasts of access windows as observation features, see Sec. 2.2.1, but such a technique has not been used in SSA sensor allocation. Inspired by the implementation in [40], this research uniquely applies forecast sensor-target access windows in the observation space of an RL agent for the SSA sensor allocation problem. This portion of the research is developed in Chs. 5 and 6.

Reward Function to Balance Global and Local Uncertainty

This research developes a reward function designed to balance global and local reduction in uncertainty. Previous research has used reward functions which either encouraged the agent to maintain a measure of uncertainty for the entire RSO catalog below a threshold [32, 48, 52] or to maximize the reduction in uncertainty among options available to the sensor [13, 14, 15, 50, 51, 53]. This research reformulates the whole-catalog metric as a continuous variable, and introduces the concept of the *custody fraction* as a reward component and a measure of effectiveness. These two components are combined with previously-used information gain metrics to form a new reward function. This portion of the research is presented in Chs. 5 and 6.

Non-Trivial Opportunities Metric

This research developes a metric for quantifying the potential for an RL agent to learn that is tailored for environments in which it is possible for an agent to have no alternatives to choose from. The fraction of *non-trivial opportunities* is the fraction of steps in an episode in which an agent has two or more valid choices to make. Such a metric, or an equivalent, is not present in the existing SSA sensor management literature. The non-trivial opportunities metric is developed in Chs. 5 and 6.

Chapter 3

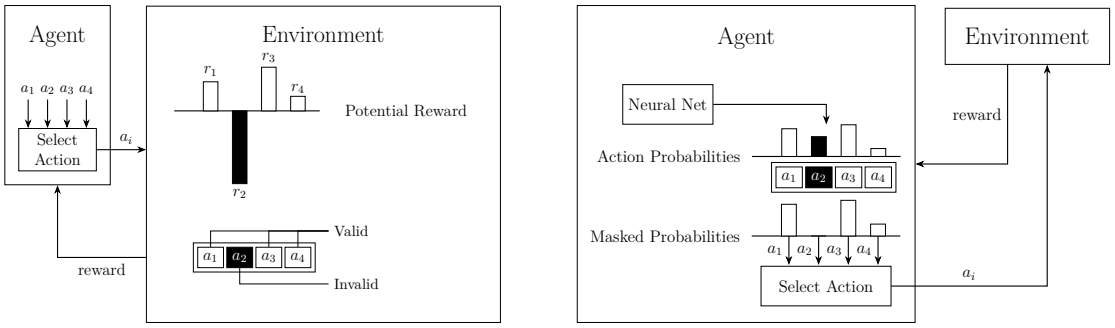
Action Masking Study Methodology

This chapter describes the methodology of the action masking study. This study compares two alternative invalid action handling techniques, penalization and masking, as applications to the SSA sensor allocation problem. Much of the methodology presented in this chapter is shared with that in Ch. 5. This chapter is based on the work conducted in [29].

3.1 Introduction

In the SSA sensor allocation problem, all sensors (terrestrial or space-based) have a limited FOR. The primary factor in limiting FOR for terrestrial sensors is the Earth, which obstructs the view of a sensor to any direction below some elevation limit driven by sensor and geography limitations (e.g., mountains). For this study, the elevation limit is assumed to be the horizon. This FOR restriction corresponds to a large portion of the RSO catalog at any given time, depending on the orbital parameters of the catalog in question. The RSO catalog is uniformly distributed and in LEO (more details in Sec. 3.2).

In many RL applications, there are portions of the action space which are invalid for subsections of the problem (e.g., in many video games, walking through a wall is not allowed). In the SSA sensor allocation problem, invalid actions can include tasking a sensor to a target which is obscured by the Earth (the use case in this study) or which is too far away for the



(a) Invalid action penalty (b) Invalid action masking

Figure 3.1: Illustrations of (a) invalid action penalty and (b) invalid action masking.

sensor to detect. In many scenarios, RL practitioners apply a penalty to the agent when it chooses an invalid action; this technique is known as *invalid action penalty*, illustrated in Fig. 3.1a [56].

Invalid action penalty allows the agent to learn which actions are invalid in the same way that it learns which actions are undesirable. However, in applications in which the action space is large and the subset of valid actions is small, invalid action penalty may slow or even stall learning by penalizing the agent from exploring the state space. The SSA sensor allocation problem is one such application.

An alternative to invalid action penalty is *invalid action masking*, illustrated in Fig. 3.1b, in which the agent is explicitly prevented from selecting invalid actions. Invalid action masking does not penalize the agent for exploration because the subspace of invalid actions is unavailable to the agent, potentially allowing the agent to learn faster relative to invalid action penalty. Because action masking is a binary operation (an action is either available or not), it is best-applied in rules-based scenarios, in which valid and invalid actions are clearly defined. Invalid action masking is more complex to implement than invalid action penalty, but is a common approach and has been shown to be effective in RL in general [56, 57, 58].

Visibility constraints between sensors and targets in the SSA sensor allocation problem

represent a rules-based constraint on the action space. A target is either inside or outside of a sensor's FOR. If the agent controlling the sensor is allowed to allocate actions (i.e., RSOs) in an unconstrained way, it is possible for the agent to task the sensor to a non-visible RSO (outside the FOR), which is an undesirable action. The desired behavior of tasking only visible targets can theoretically be learned by an RL agent. However, invalid action handling techniques have not previously been compared in the SSA sensor tasking problem. Because target visibility is a fairly firm rule (nuances are discussed later), invalid action penalty and masking are appropriate techniques to apply to an RL solution for the limited-FOR SSA sensor allocation problem.

To compare the two invalid action handling techniques shown in Fig. 3.1, an SSA environment model is developed. The SSA environment interacts with an agent via the standard RL paradigm of returning observations and rewards in response to actions [17]. The reward function used in the environment penalizes the agent for tasking non-visible RSOs. An action mask-enabled NN is developed for this study. The RL policies are trained using PPO, a popular training algorithm that has demonstrated success in a wide variety of applications [36]. Two benchmark policies are designed and analyzed for comparison to invalid action penalty and masking.

The remainder of this chapter is outlined as follows. Section 3.2 describes the SSA environment model. Section 3.3 developes reward functions for the invalid action penalty and masking agents. Section 3.5 described the implementation technique for an action masking-enabled RL agent. Section 3.6 describes training technique used in this study. Section 3.7 developes benchmark policies to provide context for the RL policies' performance. Section 3.8 describes the simulation methodology used to evaluate the performance of all policies. Section 3.9 describes the metrics used to evaluate the performance of the policies.

3.2 Space Situational Awareness Environment

A flexible SSA environment model was developed using the Gymnasium framework, which allows for a configurable number of sensors and targets that can be placed terrestrially or in orbit [59]. The environment source code is available at `github.com/dylan906`. For this study, all sensors are terrestrial and Earth-fixed, and all targets (RSOs) are in orbit. RSO dynamics are propagated using two-body techniques. Sensors are controlled by a central SSA agent. Observations are made, rewards generated, and dynamics propagated at a fixed time step. The dynamic state of each target is estimated by a UKF. Because the intent of the study is to examine sensor allocation, rather than to model sensor fidelity, generic sensors that generate full state measurements are used. Sensor measurements are the full dynamic state (position and velocity) of the RSOs.

Because the intent of this study is to examine the tasking problem, targets are assigned by the network to sensors at a relatively large time step, assuming that the sensors individually determine lower-level scheduling parameters such as pointing direction and exposure time. Real-world sensors generally have terrain-induced azimuth and elevation restrictions (e.g., mountains, buildings). The sensors in this study have no azimuth or elevation viewing restrictions, other than the horizon. The Earth is modelled as a sphere. Real-world sensors can observe multiple targets simultaneously (e.g., if multiple satellites are in the FOV of a telescope) [7]. This study does not consider serendipitous observations; each sensor observes a single RSO at a time.

In the physical world there is benefit to tasking multiple sensors to the same target (e.i., diversity of geometry). In this study, to simplify the environment, multi-sensor imaging of a single target is not modelled. When a sensor is tasked to a target, a single measurement of the target is generated. This behavior is identical even if multiple sensors are tasked to the

Table 3.1: Environment parameters (R_{E}: Earth radius, SMA: semi-major axis, RAAN: right-ascension of ascending node, AL: argument of latitude).

Parameter		**Value**
Time step		100 sec
Num. Sensors M		3
Sensor locations		[(0, 0), (0, 45), (45, 0)] (°N, °E)
Num. RSOs N		100
Horizon T		288 steps
	Orbital Parameter	**Value**
RSO distribution	SMA	$U(R_{\mathrm{E}} + 400, R_{\mathrm{E}} + 800)$ km
	inclination	$U(0, \pi/2)$ rad
	eccentricity	0
	RAAN	$U(0, 2\pi)$ rad
	AL	$U(0, 2\pi)$ rad
	UKF Parameter	**Value**
UKF	Process noise Q	$(1 \times 10^{-3})I_6[1, 1, 1, 1 \times 10^{-2}, 1 \times 10^{-2}, 1 \times 10^{-2}]^T$
	Measurement noise R	$0_{6\times6}$
	Initial covariance P_0	$10I_6[1, 1, 1, 1 \times 10^{-2}, 1 \times 10^{-2}, 1 \times 10^{-2}]^T$

same target simultaneously. The SSA environment parameters used in this study are shown in Table 3.1.

For this study, a small number of sensors M are located such that targets have long periods of time in which no sensor has access to them. The target catalog is made up of N uniformly distributed RSOs in LEO, all with circular orbits. The horizon T (the number of steps in an episode) and time step are set to simulate about eight hours per episode, allowing for many orbits of the RSOs. The UKF measurement noise R is set to zeros, to make the environment easier for the RL agent to learn. This design choice simplifies the environment, but does not affect comparative analyses between the invalid action handling techniques because all RL agents operate on the same environment.

The observation space of the environment, itemized in Table 3.2, consists of the diagonals

Table 3.2: Environment observation space (diag(P): diagonals of covariance).

Parameter	Dimensions
Covariance Diagonals diag(P)	$6N$
Discrete Estimated Visibility Map $\hat{\Omega}_d$	NM
Total number of features	$N(6+M)$

of the covariances matrices of all RSOs and the discrete estimated visibility map $\hat{\Omega}_{\mathrm{d}}$.

The discrete estimated visibility map Ω_{d} relates estimated visibility status of sensors to targets. It is an $[N, M]$-shaped binary array defined as

$$\hat{\Omega}_{\mathrm{d}} = \begin{bmatrix} v_{1,1} & \cdots & v_{1,M} \\ \vdots & \ddots & \vdots \\ v_{N,1} & \cdots & v_{N,M} \end{bmatrix}, v_{n,m} \in \{0, 1\} \tag{3.1}$$

where $v_{n,m} = 1$ indicates that the (n, m) target-sensor pair can see each other. A target is visible to a sensor provided the target is above the sensor's local horizon. There are no range restrictions.

In extreme cases, the state estimate error can be so large that the estimated and true positions yield different visibility values $v_{n,m}$. The observation space includes only the *estimated* visibility map $\hat{\Omega}_{\mathrm{d}}$, not the *true* visibility map Ω_{d}, ensuring that the SSA agent does not have unrealistic insight into the true states. If a sensor is tasked to a target that the agent believes is visible but is not, the target states are not measured.

Each sensor can choose from N targets numbered 1 to N, plus the null action, for a total of $N + 1$ available actions at each time step. The null action must be included in the action space because there may be instances where a sensor has no available targets in its FOR. There are M sensors, meaning the network action space has M dimensions. The action

vector a is defined as

$$a = [a_1, \ldots, a_M], a_m \in \{0, \ldots, N\} \tag{3.2}$$

where a_m is the action of the m'th sensor and 0 indicates the null action.

3.3 Reward Function

Reward is given to the RL agent every time step. The reward function is composed of three component reward functions: base reward, active action subsidy, and invalid action penalty. The reward function is defined as

$$r = r_{\text{base}} + r_{\text{subsidy}} + r_{\text{penalty}} \tag{3.3}$$

where the component functions are described in the remainder of this section. Table 3.3 shows the reward function parameters used in this study, the details of which are described in this and the following sections.

Table 3.3: Reward function parameters and values. Different values ε_{nv} correspond to variants of the invalid action penalty policy.

Parameter	Value
Sigmoid Function Parameter x_0	5×10^3
Sigmoid Function Parameter k	-1×10^{-5}
Active Action Subsidy Parameter ε_{aa}	0.1
Non-Visible Action Penalty Parameter ε_{nv}	{0.001, 0.01, 0.1, 0.2, 0.4, 0.6, 0.8, 1}

The mean trace of positional covariance $\bar{P}_{1:3}$ is used to quantify the catalog uncertainty, mathematically defined as

$$\bar{P}_{1:3} = \frac{1}{N} \sum_{n=1}^{N} \text{tr}(P_{1:3}^{n}) \tag{3.4}$$

where $\text{tr}(\cdot)$ is the trace function, $P_{1:3}$ denotes the diagonals of the upper-left $[3, 3]$ block of the full covariance matrix P, and the superscript n denotes the n'th target. The trace function is not included in the symbol $\bar{P}_{1:3}$ for notational brevity. A smaller value of $\bar{P}_{1:3}$ is desireable.

The base reward function is designed to encourage the RL agent to minimize the overall catalog uncertainty. At every time step, after the action has been made and observations completed, the RL agent receives a base reward r_{base} that is inversely proportional to $\bar{P}_{1:3}$. The base reward scales mean trace of positional covariance $\bar{P}_{1:3}$ to be in the range $0 \leq r_{\text{base}} \leq 1$ by

$$r_{\text{base}} = \sigma\left(\bar{P}_{1:3}\right) \tag{3.5}$$

$$\sigma(x) = \frac{1}{1 + \exp(-k(x - x_0))} \tag{3.6}$$

where $\sigma(\cdot)$ is the sigmoid function, and k and x_0 are parameters that control the steepness and x-axis location of the sigmoid function, respectively. A negative value is used for the steepness parameter k such that larger base reward r_{base} is given for smaller mean trace of covariance $\bar{P}_{1:3}$. The values of x_0 and k are shown in Table 3.3.

The base reward encourages the agent to minimize mean trace of covariance, but it is an insensitive signal. Because the mean trace of positional covariance $\bar{P}_{1:3}$ is a measure of the

whole catalog, it does not change quickly in response to individual targets being tasked or not tasked. To make the reward function prefer active actions over the null action ($a_m = 0$), the RL agent receives a subsidy for every sensor tasked to a target,

$$r_{\text{subsidy}} = \varepsilon_{\text{aa}} \sum_{m=1}^{M} f(a_m), \quad f(a_m) = \begin{cases} 0 & a_m = 0, \\ 1, & \text{otherwise} \end{cases} \tag{3.7}$$

where ε_{aa} is the active action subsidy parameter. The active action subsidy r_{subsidy} gives a reward of ε_{aa} for every sensor tasked to a target. The subsidy is bounded on $0 \leq r_{\text{subsidy}} \leq M\varepsilon_{\text{aa}}$. Both the invalid action penalty and invalid action masking policies receive the active action subsidy r_{subsidy}. The value of ε_{aa} in this study is shown in Table 3.3.

The action mask Φ is a $[(N+1), M]$-shaped binary array indicating which actions are allowed by which sensors,

$$\Phi = \begin{bmatrix} 1 & \cdots & 1 \\ & \hat{\Omega}_{\text{d}} & \end{bmatrix} \tag{3.8}$$

where a value of 1 indicates a valid action. The action mask Φ is generated from $\hat{\Omega}_{\text{d}}$ by pre-pending a row of 1's corresponding to the null action. Both the invalid action penalty and masking policies make use of the action mask Φ, but in different ways. The invalid action masking policy is detailed in Sec. 3.5.

In the invalid action penalty scheme, the agent receives a penalty for tasking sensors to non-visible targets,

$$r_{\text{penalty}} = -\varepsilon_{\text{nv}} \sum_{m=1}^{M} \left(1 - \Phi_{a_m,m}\right) \tag{3.9}$$

where ε_{nv} is the non-visible action penalty parameter and $\Phi_{a_m,m}$ is the (a_m, m) element of the action mask Φ. Equation (3.9) gives a reward of $-\varepsilon_{\text{nv}}$ for every sensor tasked to a non-visible target. The penalty is bounded by $-M\varepsilon_{\text{nv}} \leq r_{\text{penalty}} \leq 0$.

The invalid action masking policy cannot task non-visible targets, therefore $r_{\text{penalty}}(t) = 0$ for all time under that policy. The values of the reward function parameters used in this study are shown in Table 3.3. Eight different variants of invalid action penalty ε_{nv} are tested, varying the amount of the penalty, as shown in Table 3.3. The base and subsidy components of the reward function – r_{base} and r_{subsidy}, respectively, in Eq. (3.3) – are the same for all policies, and are not impacted by varying the penalty component r_{penalty}.

3.4 Action Masking

Action masking is performed on the output of the final layers of a NN, but before action probability distributions are calculated. The outputs of the final layer of a NN are called logits l, which are un-normalized probability distributions (they do not sum to one). In this study, the logits l vector is $(N+1)M$-long, where each set of $(N+1)$ elements corresponds to a different sensor. In reality, the action mask is applied over the logits vector directly, but for clarity of explanation, the logits vector is briefly treated as a 2d array.

The output logits l of the NN can be described, in this study, as an $[(N+1), M]$-shaped array, where the $+1$ term corresponds to the null action. A mask Φ is applied over the logits l by setting the elements corresponding to invalid actions to a large negative value $\varepsilon_{\text{l}} \ll 0$,

$$l^{n,m}_{\text{masked}} = \begin{cases} l^{n,m}, & \Phi_{n,m} = 1 \\ \varepsilon_{\text{l}}, & \text{otherwise} \end{cases} \tag{3.10}$$

Table 3.4: Parameters used in NN.

Parameter	Value
Logit Override Parameter ε_l	-3.4×10^{38}
FC Network Configuration	[450]
Activation Function	tanh

where $l^{n,m}$ is the (n, m) element of the logits array and ε_l is the logit override parameter. The logit override parameter ε_l is usually set to the largest negative number the programming language can represent. The value of ε_l used in this study is shown in Table 3.4. By setting the probability of selecting the invalid action to effectively zero, the agent is guaranteed to not select it.

3.5 Neural Network Architecture

The NN architecture used in this study consists of a FC network, followed by an action mask. The NN is illustrated in Fig. 3.2.

Raw observations $\mathcal{O}$, made of the elements in Table 3.2, are passed through a preprocessor before being input to the FC network. The preprocessor creates the action mask Φ via Eq. (3.8) and scales the covariance diagonals diag(P) by 1×10^{-4} to make their magnitudes near one. Rescaling observations is a common practice in RL that improves training speed and performance [60]. The covariance diagonals generally have large values that can vary by orders of magnitude. The rescaling factor was chosen heuristically. The action mask is sent to the masking block. The preprocessor flattens the scaled observations into a 1D array to interface with the FC network. The FC network's number and size of layers is configurable.

In the action masking scheme, the logits l are masked via Eq. (3.10) to produce the masked

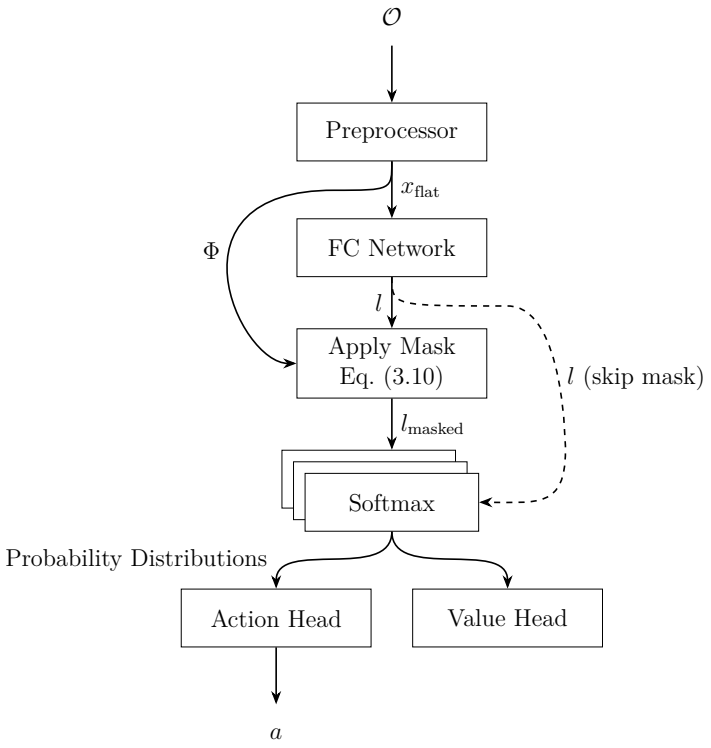


Figure 3.2: Neural network architecture used for invalid action study. Invalid action penalty scheme follows dashed line.

logits l_{masked}. In the invalid action penalty scheme, the masking step is skipped, and the logits l are sent directly to the softmax blocks, as indicated by the dashed line in Fig. 3.2.

The softmax blocks in Fig. 3.2 partition the logits vector into M sets of $(N+1)$ elements, then apply the softmax function to each set. The softmax function converts a set of arbitrarily-valued elements into a set that sums to one. The output from the softmax blocks is a set of M discrete probability distributions. The action branch accepts the probability distributions and generates the action vector a via an action selection method.

During training, actions are stochastically sampled from the probability distributions to explore the action space. Because the masked actions have effectively zero probability, they are never chosen during training. During validation, exploration is not desired, so the action associated with the highest probability, per sensor, is chosen. The value head estimates the expected value of the state-action pair, and is used to update the weights of the model during training.

The NNs trained in this study use a single-layer of neurons, the size of which was heuristically chosen, and a hyperbolic tangent (tanh) activation function, also heuristically chosen. Various hidden layer configurations and activation functions were trialed in preliminary work; the final configuration was chosen because it both is simple and produced non-trivial training results (mean episode reward responded to training) for both invalid action penalty and masking techniques. The parameters for the NN are shown in Table 3.4.

3.6 Training

The RL agent was trained with PPO. Proximal policy optimization, an algorithm known for its efficiency and adaptability to a wide variety of environments, prioritizes small policy

Table 3.5: Training parameters used in action masking study.

Parameter	**Value**
Clip parameter ϵ	0.3
Discount factor γ	0.9
Learning rate	1×10^{-5}
Training Batch Size	4000
SGD Minibatch Size	128
Training Iterations	200
Exploration Method	Stochastic Sampling

updates when learning positively by clipping the loss prior to estimating the loss function gradient [36]. Proximal policy optimization has been used successfully in the more recent RL SSA sensor allocation research, as discussed in Sec. 2.2 [14, 15, 50].

The parameters used for training are shown in Table 3.5. All policies were trained using the same number of iterations to compare performance directly. The number of training iterations was determined heuristically. Training was conducted using the Ray framework and the Virginia Tech Advanced Research Computing high performance computing cluster [49, 61]. Ray is a Python library used commonly in the industry and academia for distributed computing, particularly in ML applications.

3.7 Benchmark Policies

The trained policies are compared to two benchmark policies, called *random* and *greedy* here. This section defines the benchmark policies.

3.7.1 Random

The random policy tasks all sensors to uniformly random actions, subject to the action mask Φ. The random policy is mathematically defined as

$$\begin{aligned} a_{\text{random}} &= [a_1, \ldots, a_M] \\ a_m &\sim U(\mathcal{N}_\Phi^m) \end{aligned} \tag{3.11}$$

where $\mathcal{N}_\Phi^m$ is the set of targets visible to the mth sensor at a single time step. The random policy is disallowed to choose invalid actions. However, the null action is always valid, even when targets are visible.

3.7.2 Greedy

The greedy policy tasks each sensor to observe the RSO with the largest trace of positional covariance $\text{tr}(P_{1:3})$, subject to visibility constraints. If no targets are visible, the null action $a_m = 0$ is chosen. Mathematically, the greedy policy is

$$\begin{aligned} a_{\text{greedy}} &= [a_1, \ldots, a_M] \\ a_m &= \arg\max_x \left(\varepsilon_{\text{na}}, \text{tr}(P_{1:3}^1)\Phi_{1,m}, \ldots, \text{tr}(P_{1:3}^N)\Phi_{N,m}\right) \end{aligned} \tag{3.12}$$

where $0 < \varepsilon_{\text{na}} \ll 1$ denotes the null action subsidy, $P_{1:3}^n$ denotes the positional covariance of the nth target, and $\Phi_{n,m}$ denotes (n, m) element of the action mask. Because the value of $\Phi_{n,m}$ is either zero or one, it toggles the base value $\text{tr}(P_{1:3}^n)$ of a particular target, making the value zero when the target is not visible to the sensor.

Ties in values are resolved by a uniform random selection among tied values. In the event

where no RSOs are visible, the values of all *active* actions are zero (see Eqs. (3.1) and (3.8)). The null action subsidy ε_{na} is necessary to prevent a value tie of all zeros, which could violate the mask. Only the greedy policy uses a null action subsidy ε_{na}. The null action subsidy ε_{na} is the smallest positive number that can be represented in the programming language used.

3.8 Simulation

Policies are evaluated against each other in two simulations: a single episode experiment and a 100-episode Monte Carlo. Initial conditions are identical for all policies in the single episode experiment. In the Monte Carlo experiment, initial conditions are independent and identically distributed according to the parameters in Table 3.1. The result from the single episode experiment are representative of the more exhaustive Monte Carlo study. Results and analysis of both of these experiments are provided in Ch. 4.

3.9 Metrics

This study evaluates the results from the simulations using a few tailored metrics. This section describes those metrics.

The overall uncertainty of the catalog is quantified by the mean trace of the positional covariance, as defined in Eq. (3.4). When estimate accuracy is very poor, the covariance, and the therefore the trace of positional covariance, is large. When covariance is large, the Gaussian distribution assumption (a linearity assumption) inherent in the UKF is invalid due to the nonlinearity of the dynamics. In this situation, the covariance no longer represents a realistic physical quantification of uncertainty [7]. However, in this research, the metric $\bar{P}_{1:3}$ is not used as a physical measure, but as a general performance metric for tasking algorithms.

In this case the realism of the covariance is immaterial gauging the performance of policies. However, it is import to note so as to not interpret the results as a physical measure.

Analyzing a statistical measure of the catalog covariance gives a good indication of top level performance of the policies. However, to gain insight into the efficacy of the training process and the operational behavior of the trained policies, other metrics can also be used. This section introduced three metrics that are used to quantify the behavior of the policies: null actions, invalid actions, and wasted opportunities. All three of these metrics are an average over the sensors and time steps in an episode. A utility function that sums the sensors and time steps is used to calculate the metrics,

$$f_{\text{util}}(x_m) = \sum_{k=0}^{T} \sum_{m=0}^{M} f(x_m) \tag{3.13}$$

where T is the number of time steps, M is the number of sensors, and $f(x_m)$ is a measure (or measures) of sensor m, defined next.

The null action fraction F_{na} is the proportion of actions over an episode in which a policy chooses the null action, mathematically defined as

$$F_{\text{na}} = \frac{1}{TM} f_{\text{util}}\left(f_{\text{na}}(a_m)\right) \tag{3.14a}$$

$$f_{\text{na}}(a_m) = \begin{cases} 1, & a_m = 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.14b}$$

where a_m is the action chosen by sensor m. The null action fraction is useful for understanding a policy's behavior relative to benchmark policy. Most scenarios (i.e., catalog size, orbit

parameters) result in some portion of time steps in which a sensor has no RSOs in view. In these cases, the null action is appropriate to take. Comparing the baseline null action fraction F_{na}, from some benchmark policy, to that of a trained policy gives insight into the policy's capacity to choose active actions when available.

The invalid action fraction F_{ia} is the proportion of actions over an episode in which a policy chooses an invalid action, mathematically defined as

$$F_{\text{ia}} = \frac{1}{TM} f_{\text{util}}\left(f_{\text{ia}}(a_m, \Phi_{a_m,m})\right) \tag{3.15a}$$

$$f_{\text{ia}}(a_m, \Phi_{a_m,m}) = \begin{cases} 1, & \Phi_{a_m,m} = 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.15b}$$

where $\Phi_{a_m,m}$ is the action mask value for the (a_m, m) target-sensor pair. If the action mask is violated, meaning the sensor is tasked to a non-visible target, the value of $\Phi_{a_m,m} = 0$, and the action is counted as invalid. The invalid action fraction F_{ia} is always 0 for the benchmark policies and the action masking policy, as they are by definition disallowed from tasking sensors to observe non-visible targets. This metric is useful for gauging the effectiveness of the invalid action penalization scheme; a low value indicates that the penalty is effective at training the policy to avoid invalid actions.

Because the estimated visibility can differ between policies, the total number of non-trivial tasking opportunities in an episode can also be different. For example, Policy A may estimate no visible RSOs at the same time Policy B estimates many visible RSOs. To capture this perceived difference in opportunities, the number of non-trivial tasking opportunities N_{nt} is

defined as

$$N_{\text{nt}} = f_{\text{util}}\left(f_{\text{nt}}(\Phi_{*,m})\right) \tag{3.16a}$$

$$f_{\text{nt}}(\Phi_{*,m}) = \begin{cases} 1, & \sum \Phi_{*,m} > 1 \\ 0, & \text{otherwise} \end{cases} \tag{3.16b}$$

where $\sum \Phi_{*,m}$ is the sum of the m'th column of the action mask matrix. The first row of the action mask Φ is always one because the null action, $a_m = 0$, is always allowed. Therefore, if the sum of a column of Φ is greater than 1, then active actions are available to the sensor, and the action is counted as a non-trivial tasking opportunity. In practice, the number of non-trivial opportunities N_{nt} varies little between policies when large catalogs are used. Regardless of how large an estimate covariance is, there is usually at least one estimated-to-be visible RSO for every sensor at every time step. The number of non-trivial tasking opportunities N_{nt} is usually near or equal to the total number of actions TM.

Wasted opportunities are the subset of non-trivial opportunities in which the sensor chooses the null action. The wasted opportunity fraction F_{wo} is the proportion of opportunities over an episode in which a policy chooses the null action when at least one RSO is estimated to be visible. The wasted opportunity fraction is mathematically defined as

$$F_{\text{wo}} = \frac{1}{N_{\text{nt}}} f_{\text{util}}\left(f_{\text{wo}}(a_m, \Phi_{*,m})\right) \tag{3.17a}$$

$$f_{\text{wo}}(\Phi_{*,m}, a_m) = \begin{cases} 1, & \sum \Phi_{*,m} > 1 \quad \text{and} \quad a_m = 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.17b}$$

where $a_m = 0$ is sensor m being tasked to the null action. The wasted opportunity fraction F_{wo} is always zero for the greedy policy because it chooses the null action only when no RSOs are estimated to be visible. In this study, such incidences happen rarely. All other policies can have a non-zero wasted opportunity fraction F_{wo}.

Chapter 4

Action Masking Study Results

This chapter presents the results and analysis from the action masking study, the methodology of which is in Ch. 3. The results and analysis are presented in the following sections. Section 4.1 presents the results from the RL training process. Section 4.2 analyzes the results a single episode implementing the trained and benchmark policies. Section 4.3 analyzes the results from a Monte Carlo simulation of the policies. This chapter is based on the work conducted in [29].

4.1 Training Analysis

Two types of RL policies were trained on the SSA sensor allocation problem defined in Ch. 3: invalid action masking and invalid action penalty. The invalid action masking policy is explicitly prevented from tasking sensors to non-visible targets. Invalid action penalization policies are discouraged from selecting non-visible targets by a penalty term in the reward function, which punishes the RL agent when a non-visible target is selected. The goal of the penalization scheme is to train the policies to entirely avoid invalid actions. Eight penalization policies were trained, each with a different magnitude of penalty. This section presents and analyzes the results from the training process.

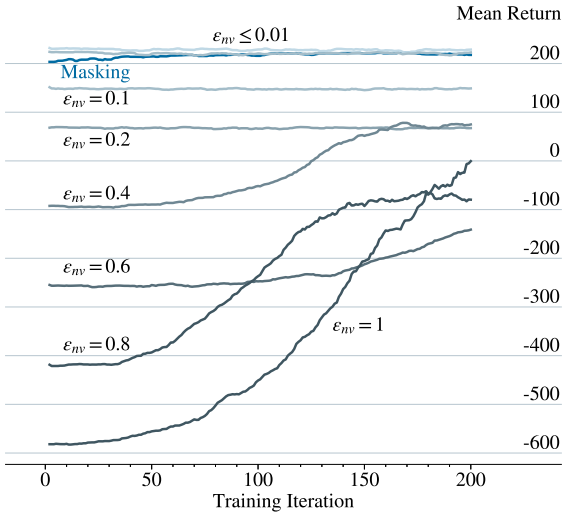
The training curves for all policies are shown in Fig. 4.1 in terms of mean return (Fig. 4.1a), delta mean return (Fig. 4.1b), and Kullback–Leibler (KL) divergence (Fig. 4.1c). Figure 4.1a

shows the return (cumulative reward at the end of an episode) of the policy at each training iteration, averaged among all episodes during the training iteration. The number of episodes per iteration varies, and is generally around 10 – 20. A policy's learning success can be qualitatively determined by observing if the mean return (Fig. 4.1a) increases over time before plateauing. Figure 4.1b shows the mean delta return (change in return relative to initial iteration). Figure 4.1c shows the KL divergence over training iterations. The KL divergence is a measure of how much the action probability distribution of the current iteration differs from the previous. A pair of distributions with a higher value of KL divergence are more statistically different than a pair corresponding to a lower value.

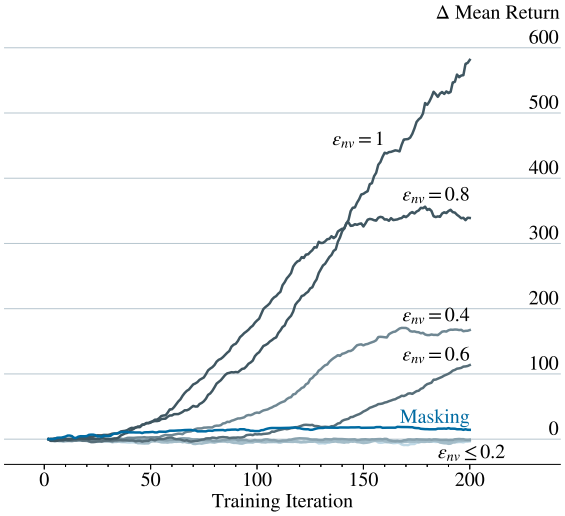
In Figs. 4.1a and 4.1b, mean episode return remains flat over training for the penalization policies with $\varepsilon_{\text{nv}} < 0.4$. For penalization policies with $\varepsilon_{\text{nv}} \geq 0.4$, the amount of growth in mean return increases as penalty increases. However, increase in mean return does not necessarily equate to better performance relative to other policies, only that an agent is learning. For example, the $\varepsilon_{\text{nv}} = 1$ policy, which has the largest return growth, finishes training with less mean return than all but two policies.

The masking policy has slight growth in mean return over training, indicating that the agent learned to perform better over time. This mean return growth occurs early in training, and plateaus around iteration 50.

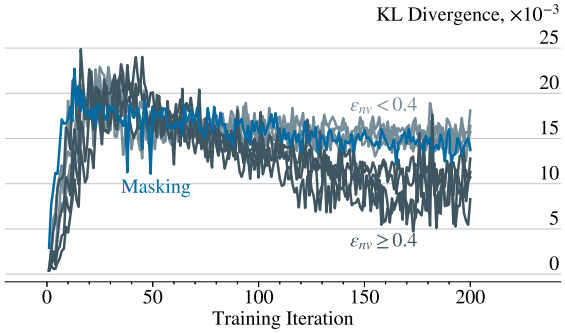
As the magnitude of the penalty component of the reward function r_{penalty} decreases, the reward functions for the masking and penalization policies become more similar, see Eq. (3.3). The masking policy starts with an inherent reward advantage in that it cannot be penalized for choosing invalid actions. Therefore, the reward floor for the masking policy is higher than that of the penalization policies. In turn, lower reward *growth* for the masking policy, relative to the other policies, is expected. In terms of *final* mean return, the reward ceiling, the masking and $\varepsilon_{\text{nv}} = \{0.01, 0.001\}$ policies are about equivalent. The $\varepsilon_{\text{nv}} = \{0.1, 0.2\}$



(a) Training progress increases with increased penalty value ε_{nv}. The masking policy and $\varepsilon_{\text{nv}} = \{0.01, 0.001\}$ overlap most of training.



(b) Same data as (a), but each series is debiased by its first value. The policieis with $\varepsilon_{\text{nv}} \leq 0.2$ closely overlap.



(c) Policies colored by behavior. The masking policy increases slightly faster than the other policies.

Figure 4.1: Training progress for the action masking policy and invalid action penalty policies (varying ε_{nv}). The masking policy and small-values penalty policies train similarly in (a) mean return, (b) delta mean return, and (c) KL divergence.

policies, like the $\varepsilon_{\text{nv}} = \{0.01, 0.001\}$ policies, do not show return growth over training, and have lower overall return, as seen in the upper portion of Fig. 4.1a and lower portion of Fig. 4.1b.

The KL divergence of the penalization policies can be characterized in two groups: those with $\varepsilon_{\text{nv}} < 0.4$ and otherwise. The KL divergence (Fig. 4.1c) of the $\varepsilon_{\text{nv}} < 0.4$ policies increases initially, then levels out after about 25 iterations, indicating that action probability distributions do not significantly change thereafter. The KL of the penalization policies with the larger reward, $\varepsilon_{\text{nv}} \geq 0.4$, have the same initial increase, but then decrease to less than the other policies around iteration 100 and continue to decrease until the end of training. The low KL divergence in the $\varepsilon_{\text{nv}} \geq 0.4$ policies indicates that the action probability distributions converge toward a more deterministic solution (less stochastic). This convergence is not necessarily indicative of quality, but shows that the policies are training.

The KL divergence of the masked policy stabilizes faster than that of the penalization policies. The masked policy maintains a slightly lower KL divergence than the $\varepsilon_{\text{nv}} < 0.4$ policies during the plateau portion of training. This is a result of the masked policy having many of its action distribution bins set to zero from masking. With many bins already set to zero, changes in the non-zero probability bins corresponds to a greater change in KL distribution. That the masking policy's KL distribution quickly increases, then stabilizes, shows that learning is occurring early in training. The KL data corroborates the analysis of the mean episode reward data: the masking policy learns early, and the $\varepsilon_{\text{nv}} \geq 0.4$ learns the most.

4.2 Single Episode

All policies (one action masking, eight penalization, and two benchmark policies) were run through a single episode (simulation) with identical initial conditions. The initial condi-

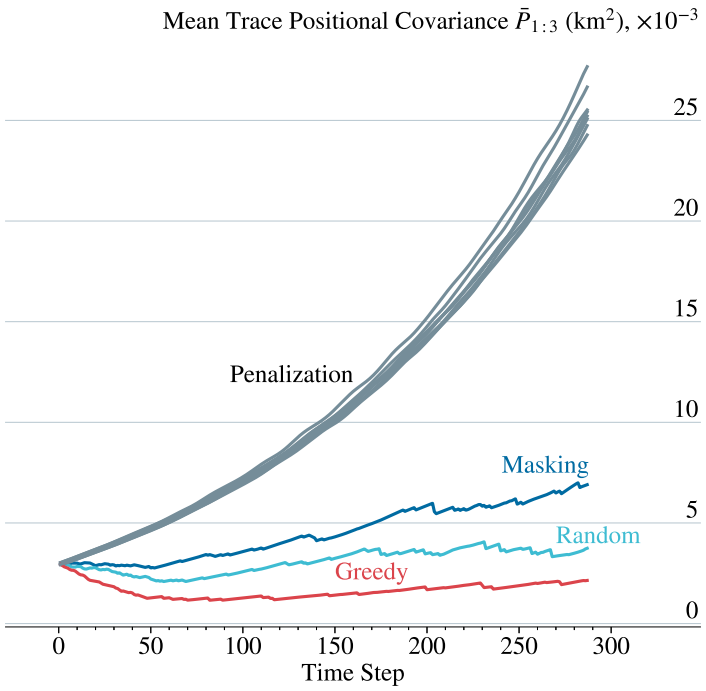


Figure 4.2: Mean trace of positional covariance $\bar{P}_{1:3}$ from a single episode (lower is better). The penalization policies closely overlap.

tions for the single episode experiment are in Table 3.1. This method allows a one-to-one comparison of policy performance on identical RSO catalogs.

The performance of the policies is evaluated in terms of the mean trace of positional covariance $\bar{P}_{1:3}$ and action allocation behaviors, which are defined in Sec. 3.9.

4.2.1 Uncertainty

The mean trace of positional covariance $\bar{P}_{1:3}$, defined in Eq. (3.4), from the single episode experiment is shown in Fig. 4.2. A higher value indicates that the RSO catalog has on average greater uncertainty.

In Fig. 4.2, the invalid action penalization policies closely overlap each other, with little

distinction in terms of $\bar{P}_{1:3}$. This results indicates that the penalization policies are tasking invalid actions (non-visible targets) much of the time, selecting the null action much of the time, or are tasking select targets in such a way as to not reduce the mean uncertainty of the catalog. The action allocation behavior of the penalization policies is examined in Sec. 4.2.2. The penalization policies end the episode with the highest $\bar{P}_{1:3}$ of the policies. There are no distinguishable points in the episode in which the penalization policies reduced the mean uncertainty of the catalog.

The invalid action masking policy has a lower $\bar{P}_{1:3}$ than all penalization policies, and higher than the benchmark policies. As expected, the random policy has a greater average uncertainty than the greedy policy. The greedy policy is designed to task targets with high $P_{1:3}$, so it should have a smaller $\bar{P}_{1:3}$ than the random policy, which chooses targets arbitrarily. The masking, random, and greedy policies all exhibit decreases in $\bar{P}_{1:3}$ throughout the episode, indicating that the policies are tasking targets with high enough uncertainty so as to noticeably reduce the mean uncertainty of the catalog.

4.2.2 Action Allocation

Analyzing the policies in terms of uncertainty $\bar{P}_{1:3}$, as in Sec. 4.2.1, gives a high-level assessment of performance. However, in order to to understand the underlying behavior of the policies, a more detailed examination of how policies allocate actions is necessary. This section examines the action allocation metrics, discussed in Sec. 3.9, of the policies from the single episode experiment.

The action allocation metrics null action fraction F_{na}, invalid action fraction F_{ia}, and wasted opportunity fraction F_{wo}, along with the final mean trace of positional covariance $\bar{P}_{1:3}(T)$ from the single episode experiment are shown in Table 4.1. See Sec. 3.9 for definitions of

Table 4.1: Action allocation metrics of the single episode experiment. Number of actions $TM = 864$. Number of non-trivial opportunities N_{nt} varies. Lower is better for $\bar{P}_{1:3}$, F_{ia}, and F_{wo}.

Policy	$\bar{\boldsymbol{P}}_{\mathbf{1:3}}(\boldsymbol{T})$, **km**^{2}	**Null Actions,** $\boldsymbol{F}_{\text{na}}$ %	**Invalid Actions,** $\boldsymbol{F}_{\text{ia}}$ %	**Wasted Opportunities,** $\boldsymbol{F}_{\text{wo}}$ %
Greedy	2.14×10^3	2.1	0.0	0.0
Random	3.74×10^3	28.9	0.0	25.8
Masking	6.90×10^3	7.1	0.0	5.1
$\varepsilon_{\text{nv}} = 10^{-3}$	2.67×10^4	0.0	92.8	0.0
$\varepsilon_{\text{nv}} = 10^{-2}$	2.55×10^4	0.0	94.7	0.0
$\varepsilon_{\text{nv}} = 10^{-1}$	2.47×10^4	0.0	95.4	0.0
$\varepsilon_{\text{nv}} = 2 \times 10^{-1}$	2.54×10^4	33.2	61.8	33.2
$\varepsilon_{\text{nv}} = 4 \times 10^{-1}$	2.77×10^4	100.0	0.0	100.0
$\varepsilon_{\text{nv}} = 6 \times 10^{-1}$	2.51×10^4	33.3	62.0	33.3
$\varepsilon_{\text{nv}} = 8 \times 10^{-1}$	2.43×10^4	66.7	28.8	66.4
$\varepsilon_{\text{nv}} = 1$	2.52×10^4	100.0	0.0	100.0

the metrics. The action allocation metrics in Table 4.1 are shown as percents ($100F_i$). The number of actions TM, used in calculating F_{na} and F_{ia}, is constant for all policies. The number of non-trivial opportunities N_{nt}, used in calculating F_{wo}, varies slightly between trials; see discussion around Eq. (3.16a) for details. The values for $\bar{P}_{1:3}(T)$ in Table 4.1 are the same as the final points in Fig. 4.2.

In Table 4.1, the greedy policy chooses null actions only when no RSOs are estimated to be visible. The null action fraction F_{na} of the greedy policy in Table 4.1 represents an approximate lower bound of the scenario. By design, the greedy policy does not log any wasted opportunities. Likewise, the benchmark (i.e., random and greedy) and action masking policies cannot choose invalid actions, so their values for invalid action fraction F_{ia} are 0. The action masking policy's null action fraction F_{na} value is in between those of the greedy and random policies (closer to that of greedy). Similar to null actions, the masking policy's

wasted opportunity fraction F_{wo} is in-between those of the greedy and random policies. The action masking policy chooses active actions more frequently than random, which is a positive indicator that training was successful.

The maximum penalty variant of the penalization policies ($\varepsilon_{\text{nv}} = 1$) chose inaction every time step – the high penalty beat the agent into submission. The agent perceives (incorrectly) inaction to be the most desireable action at every opportunity. The low-valued penalization policies ($\varepsilon_{\text{nv}} \leq 0.1$) tasked invalid actions most of the time, as seen in Table 4.1. This indicates that the penalty in the low-valued penalization policies is not high enough to train the agents to avoid invalid actions. The reallocation of actions from mostly-null to mostly-invalid, as penalty ε_{nv} decreases from 1 to 0.1, indicates that there is a transition region of penalties that may result in more effective behavior.

This transition region is in the policies with middle-ranged penalty values of $0.2 \leq \varepsilon_{\text{nv}} \leq 0.8$. The policies with penalties below this range choose invalid actions over 90% of the time, whereas the policy above this range chooses the null action 100% of the time. The transition policies switch between these two extremes, but not smoothly. As the penalty decreases, the null action fraction F_{na} decreases, then increases back to 100% again, then decreases. This sharp change in behavior with penalty value indicates that the invalid penalty scheme is highly, nonlinearly sensitive to the penalty value ε_{nv}. Nonlinear and chaotic behavior is commonly seen in invalid action penalization; tuning the penalty value is often impractical [56].

Despite the action masking policy having fewer wasted opportunities F_{wo} than that of the random policy, action masking ended the episode with a higher mean trace of positional covariance $\bar{P}_{1:3}(T)$. This phenomenon indicates that, despite the masking policy choosing active actions more often than the random policy, when the masking policy selects targets, it chooses them such that the mean uncertainty of the system is higher than that of the random

policy. The action masking agent tasked RSOs that did not necessarily need measurements, neglecting those with higher uncertainty, and resulting in an overall higher $\bar{P}_{1:3}$.

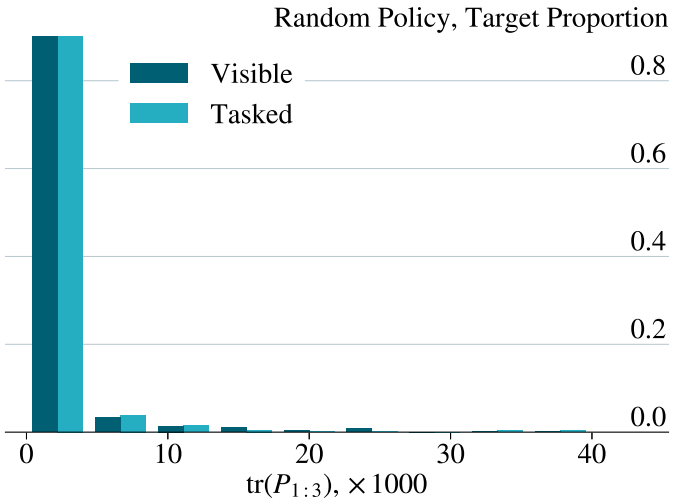
4.2.3 Visible vs. Tasked Behavior

Table 4.1 shows that the masking policy has a lower wasted opportunity fraction F_{wo} than the random policy, yet a higher (more uncertain) final mean trace of positional covariance $\bar{P}_{1:3}(T)$. The disparity in F_{wo} and $\bar{P}_{1:3}(T)$ indicates an inefficiency in how the action masking policy is distributing taskings. To gain more insight into how the masking policy is allocating taskings, the distribution of taskings, as binned by $\text{tr}(P_{1:3})$, is examined.

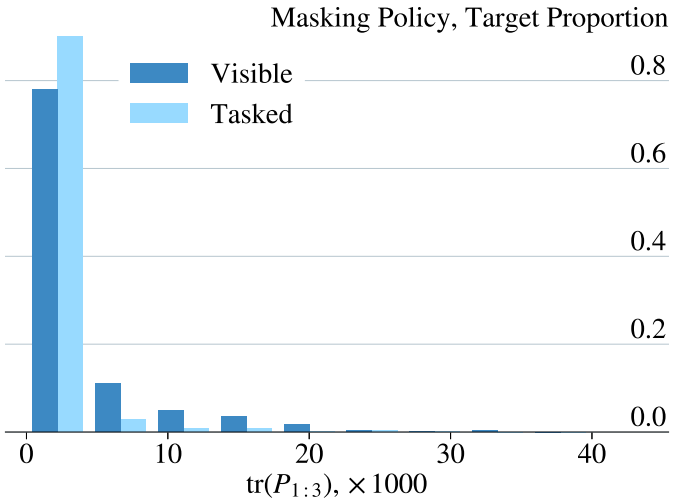
For every sensor m at every time step t, a set $\mathcal{N}_{\text{vis}}^{m}(t)$ is created that consists of the $\text{tr}(P_{1:3})$ value for every RSO that is visible to the sensor at that time, regardless of which RSO was chosen. If a target is visible to multiple sensors, then that target appears in both sensors' sets. The sensor sets are combined per time step as $\mathcal{N}_{\text{vis}}(t) = \{\mathcal{N}_{\text{vis}}^{1}(t), \ldots, \mathcal{N}_{\text{vis}}^{M}(t)\}$. The sets for every time step $\mathcal{N}_{\text{vis}}(t)$ are combined to create a set $\mathcal{N}_{\text{vis}} = \{\mathcal{N}_{\text{vis}}(0), \ldots, \mathcal{N}_{\text{vis}}(T)\}$ of $\text{tr}(P_{1:3})$ values for all visible RSOs for all time. This process is repeated again, but including only tasked RSOs, resulting in a second set $\mathcal{N}_{\text{tasked}}$. In both the visible and tasked cases, instances in which there are no RSOs visible to a sensor are ignored. The size of the visible set $|\mathcal{N}_{\text{vis}}|$ is larger than that of the tasked set $|\mathcal{N}_{\text{tasked}}|$ because the visible set counts all targets visible to the sensor network at every step, whereas the tasked set counts only those that were tasked, which is limited to no greater than M per time step.

The distributions $\mathcal{N}_{\text{vis}}$ and $\mathcal{N}_{\text{tasked}}$ for the random and masking policies, taken from the single episode experiment, are shown in Fig. 4.3. The distributions are normalized to their respective totals to better compare the visible to the tasked distributions.

Each plot in Fig. 4.3 shows the proportion of targets, binned by $\text{tr}(P_{1:3})$, during the single



(a) Random policy ($|\mathcal{N}_{\text{vis}}| = 3184$, $|\mathcal{N}_{\text{tasked}}| = 614$)



(b) Masking policy ($|\mathcal{N}_{\text{vis}}| = 3678$, $|\mathcal{N}_{\text{tasked}}| = 803$)

Figure 4.3: Distribution of $\text{tr}(P_{1:3})$ for (a) random and (b) masking policies for single episode experiment. Targets are grouped by those visible to a sensor $\mathcal{N}_{\text{vis}}$ and those a sensor was tasked to observe $\mathcal{N}_{\text{tasked}}$. Distributions are normalized to their respective totals.

episode experiment. Targets appear most often in the lowest covariance bin because, during the steps immediately following a tasking event, a recently-tasked target has a low $\mathrm{tr}(P_{1:3})$. Usually any particular RSO is visible to a sensor for multiple sequential steps. If the RSO is tasked at any step prior to the final one of an observation window, multiple steps of low uncertainty occur. This increases the count of low-uncertainty targets in the visible distributions in Fig. 4.3.

As anticipated, the random policy tasks targets in proportion to how often the targets are visible, across the full range of $\mathrm{tr}(P_{1:3})$ bins. The random policy is equally likely to task any visible target, regardless of its uncertainty. The masking policy favored targets with lower covariance, as can be seen in Fig. 4.3b by tasked distribution's higher count in the lowest covariance bin, relative to the visible distribution, and corresponding lower counts in the other covariance bins. The masking policy tasks targets with higher $\mathrm{tr}(P_{1:3})$ (those that "need" more observations) at a relatively lower rate than those with a lower $\mathrm{tr}(P_{1:3})$.

4.2.4 Wasted Opportunities

That the masking policy has non-zero wasted opportunities ($F_{\mathrm{wo}} > 0$), as shown in Table 4.1, is peculiar given that the agent receives a small reward for simply choosing an active action over the null action, see Eq. (3.7). Conversely, ML training processes are stochastic by design, so it is not necessarily unexpected that unpredictable traits would be trained into a model [62]. Insight into the masking policy's behavior can be gained by examining the distribution of wasted opportunities as compared to active actions.

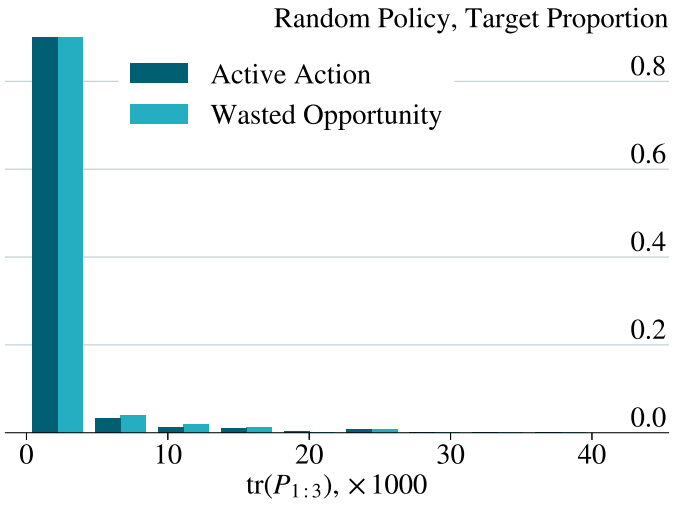
Wasted opportunities can be attributed to either the stochasticity of the RL policy or to some other systemic property of the system (e.g., reward function design, initial conditions). This section analyzes the statistical occurrence of wasted opportunities to determine if the

behavior can be attributed to inherent policy stochasticity. The objectives of analyzing the statistical occurrence of wasted opportunities is to determine if such events are correlated with a particular region of the observation space (values of $\text{tr}(P_{1:3})$). If wasted opportunities are overly-represented in a specific range of observation states, then one could conclude that such behavior is *not* attributable to the uniform noise inherent in the RL policy, but is instead systemic to the reward function.

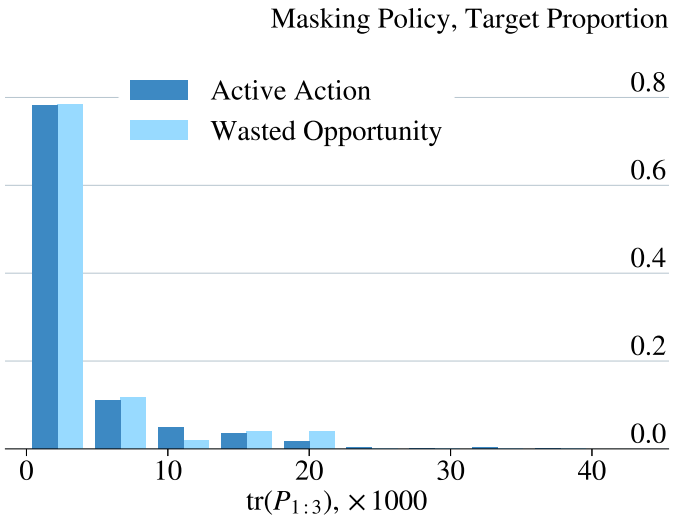
Similar to Sec. 4.2.3, the policy's behavior is characterized in terms of the trace of positional covariance $\text{tr}(P_{1:3})$. To determine if wasted opportunities are overly-represented in a range of uncertainty bins, a baseline is needed to compare against. The opposite of a wasted opportunity is an active action, when a sensor tasks an available target. The distributions of wasted opportunities $\mathcal{N}_{\text{wasted}}$ and active actions $\mathcal{N}_{\text{active}}$ are compared to determine if the wasted opportunities are out-of-family. The set of active actions $\mathcal{N}_{\text{active}}$ corresponds to all RSOs that are visible during an active action, regardless of which target is tasked. The set of wasted opportunities $\mathcal{N}_{\text{wasted}}$ corresponds to all RSOs that are visible during when a sensor is assigned the null action while at least one target is visible. Instances in which there are no RSOs visible to a sensor are ignored.

The distributions of both the sets of uncertainty values ($\mathcal{N}_{\text{active}}$ and $\mathcal{N}_{\text{wasted}}$) are shown in Fig. 4.4. The random and masking policies are shown in Figs. 4.4a and 4.4b, respectively. The distributions are normalized to their respective totals.

In Fig. 4.4, the distributions of active actions and wasted opportunities are nearly identical between each other. This indicates that the covariance of a target does not impact how either the random or masking policies select active actions vice waste opportunities. The distributions are not identical when comparing the random and greedy policies. The masking policy (Fig. 4.4b) has more weight in the higher uncertainty bins than the random policy (Fig. 4.4a). This is because the random policy has generally lower catalog uncertainty



(a) Random policy ($|\mathcal{N}_{\text{active}}| = 2505$, $|\mathcal{N}_{\text{wasted}}| = 618$)



(b) Masking policy ($|\mathcal{N}_{\text{active}}| = 3436$, $|\mathcal{N}_{\text{wasted}}| = 51$)

Figure 4.4: Distribution of $\text{tr}(P_{1:3})$ for (a) random and (b) masking policies for single episode experiment. Targets are grouped by those selected by the agent for observation $\mathcal{N}_{\text{active}}$ and those available but not selected $\mathcal{N}_{\text{wasted}}$. Distributions are normalized to their respective totals.

(smaller $\mathrm{tr}(P_{1:3})$) throughout the episode than the masking policy, as seen in Fig. 4.2. The difference in distributions between the random (Fig. 4.4a) and masking (Fig. 4.4b) policies is an artifact of general policy behavior, not specifically related to how the policies select active vice null actions.

In Fig. 4.4a, the random policy distributes active actions and wasted opportunities nearly identically to each other, which is as expected for a random policy. The same characteristic is seen in the masking policy, Fig. 4.4b. Therefore, the fact that the RL-based masking policy has a non-zero wasted opportunities fraction F_{wo} can be attributed to the stochastic nature of the policy, and not some other systemic property.

4.3 Monte Carlo

The Monte Carlo experiment consists of running each policy, those trained in Sec. 4.1 and the benchmark policies, over 100 episodes with independent and identically distributed initial conditions. Details of the simulation configuration are in Sec. 3.8. This analysis focuses on the final time step of each episode (unlike Sec. 4.2, which analyzed all steps of a single episode).

4.3.1 Uncertainty

The distributions of the trace of positional covariance at the final time step $\mathrm{tr}(P_{1:3}(T))$ are shown in Fig. 4.5. The distributions are unimodal, so box-and-whisker plots are used to represent the data in Fig. 4.5. Each box spans the first through third quartile, the whiskers span 1.5 times the inter-quartile range (IQR), and each black line shows the median. Outliers are not shown in Fig. 4.5 for clarity. All policies exhibited a few outliers, none of which

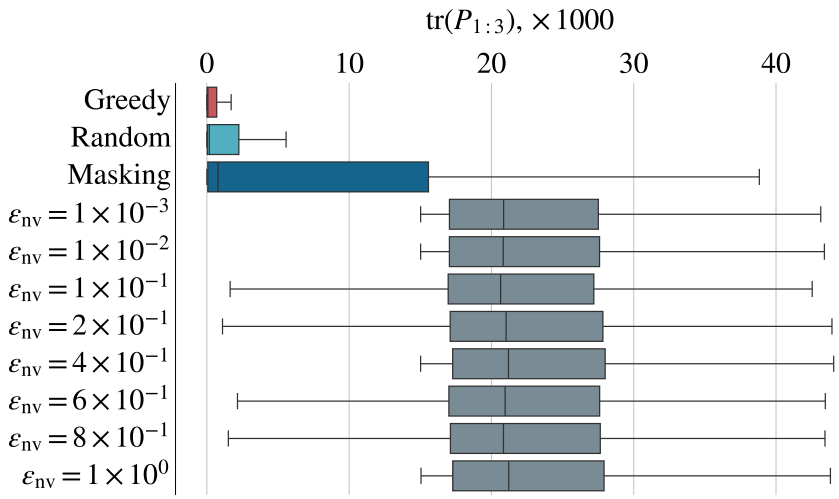


Figure 4.5: Distribution of $\mathrm{tr}(P_{1:3}(T))$ for Monte Carlo experiment (lower is better; 100 episodes).

differentiated performance.

In Fig. 4.5, the policies stratify into two groups: those with low and high uncertainty $\mathrm{tr}(P_{1:3})$. The action masking and benchmark policies exhibit distinctly lower median uncertainty than the penalization policies. As seen in the single episode experiment (Sec. 4.2), the random policy has, on median, lower $\mathrm{tr}(P_{1:3})$ than the action masking policy, and higher $\mathrm{tr}(P_{1:3})$ than the greedy policy. Action masking has the largest median uncertainty among the higher-performers. A zoomed in version of Fig. 4.5 that shows only the benchmark and masking policies is shown in appendix B.2.

The greedy, random, and masking policies all have distributions concentrated around small values of $\mathrm{tr}(P_{1:3})$. As seen in Sec. 4.2, this is a result of successive taskings, where targets that are recently tasked have small covariances. The random and masking distributions are skewed right with long tails because a small number of targets' covariances have become very large due to lack of tasking. The same characteristic is also in the greedy distribution, although less visible in Fig. 4.5.

The masking policy has a larger IQR of $\mathrm{tr}(P_{1:3})$ than the random policy, which in turn has a larger IQR than the greedy policy. It is expected that the random policy would have a greater spread of uncertainty than the greedy policy because the greedy policy more purposefully tasks for high-uncertainty than the random policy. The fact that the masking policy's spread of uncertainty is significantly larger than the benchmarks indicates that the catalog has a wide range of uncertainties at the end of an episode. In Fig. 4.3b, the masking policy is shown to favor targets with a low $\mathrm{tr}(P_{1:3})$. In Fig. 4.5, the masking policy's median $\mathrm{tr}(P_{1:3})$ is much lower than its third quartile, in fact on the order of the random policy's median. This result indicates that the masking policy favors tasking low-uncertainty targets, leaving medium- and high-uncertainty targets' covariance to degrade. This behavior is supports the single episode results in Sec. 4.2.

The penalty policies performed equivalently to each other, in terms of both median and spread of $\mathrm{tr}(P_{1:3})$. The performance of the invalid action penalty policies indicates that the behavior observed in the single episode experiment, of tending to choose the null action, holds over a wide range of initial conditions.

4.3.2 Action Allocation

Similar to Sec. 4.2.2 for a single episode, this section analyzes the how the various policies allocate actions in the Monte Carlo experiment. The action allocation metrics, null action fraction F_{na}, invalid action fraction F_{ia}, and wasted opportunity fraction F_{wo} from the Monte Carlo experiment are shown in Table 4.2. The median of each metric is taken with respect to the final time step in all episodes in the Monte Carlo simulation. The median of mean trace of final positional covariance $\bar{P}_{1:3}(T)$ is also shown; these are the same values as in Fig. 4.5. The medians of end-of-episode metrics are shown for all policies in Table 4.2.

Table 4.2: Medians of end-of-episode action allocation metrics from Monte Carlo simulations. Number of episodes = 100. Number of actions per episode $TM = 864$. Number of non-trivial opportunities N_{nt} varies. Lower is better for $\bar{P}_{1:3}$, F_{ia}, and F_{wo}.

	Median			
Policy	$\bar{\boldsymbol{P}}_{\mathbf{1:3}}(\boldsymbol{T})$**, km**^{2}	**Null Actions,** $\boldsymbol{F}_{\text{na}}$ %	**Invalid Actions,** $\boldsymbol{F}_{\text{ia}}$ %	**Wasted Opportunities,** $\boldsymbol{F}_{\text{wo}}$ %
Greedy	3.31×10^3	3.2	0.0	0.0
Random	4.98×10^3	27.5	0.0	24.7
Masking	7.58×10^3	10.5	0.0	6.6
$\varepsilon_{\text{nv}} = 1 \times 10^{-3}$	2.39×10^4	0.0	95.0	0.0
$\varepsilon_{\text{nv}} = 1 \times 10^{-2}$	2.37×10^4	0.0	94.8	0.0
$\varepsilon_{\text{nv}} = 1 \times 10^{-1}$	2.37×10^4	0.0	95.1	0.0
$\varepsilon_{\text{nv}} = 2 \times 10^{-1}$	2.40×10^4	33.2	63.7	33.3
$\varepsilon_{\text{nv}} = 4 \times 10^{-1}$	2.45×10^4	100.0	0.0	100.0
$\varepsilon_{\text{nv}} = 6 \times 10^{-1}$	2.39×10^4	33.3	63.8	33.5
$\varepsilon_{\text{nv}} = 8 \times 10^{-1}$	2.40×10^4	66.7	31.7	66.5
$\varepsilon_{\text{nv}} = 1$	2.46×10^4	100.0	0.0	100.0

The observations from the single episode experiment in Table 4.1 are generally repeated in Table 4.2. The greedy, random, and masking policies do not select any invalid actions, as designed. Also as designed, the greedy policy does not have any wasted opportunities. The masking policy's median null action fraction F_{na} is in-between those of the greedy and random policies, indicating that the trained policy makes active tasking decisions less often than random, but more than the scenario's "floor". The median wasted opportunities fraction F_{wo} of the masking policy is lower than that of the random policy, a desired characteristic.

The penalization policies can be divided into two groups: those that always chose active actions ($\varepsilon_{\text{nv}} = \{0.001, 0.01, 0.1\}$) and others. The always-active group does not select any null actions ($F_{\text{na}} = 0$), but chooses invalid actions most of the time ($F_{\text{na}} \simeq 0.95$). This indicates that the penalty values ε_{nv} are not large enough to train the again to task only visible targets.

The other group ($\varepsilon_{\text{nv}} = [0.2, 0.4, 0.6, 0.8, 1]$) allocated actions in a more varied way. Two of the policies, $\varepsilon_{\text{nv}} = \{0.2, 0.6\}$, allocate about a third of their actions to null, and the remainder to invalid actions. The $\varepsilon_{\text{nv}} = 0.8$ policy reverses these allocations, with about two-thirds of actions being null, and the remainder invalid. The $\varepsilon_{\text{nv}} = \{0.4, 1\}$ policies allocate all actions to null. The large steps in action allocation behavior between penalty values indicates that the training process is highly sensitive to the penalty value ε_{nv}. The fact that the change in allocations is not linear with respect to penalty value (see the $\varepsilon_{\text{nv}} = 0.4$ policy) indicates that the training process is also nonlinearly influenced by the penalty value, which is a known phenomenon in ML. Although the penalization policies perform equivalently in terms of uncertainty $\bar{P}_{1:3}(T)$, the ways in which they allocate actions is varied.

Overall, the Monte Carlo results support the findings in the single episode experiment.

4.4 Conclusions

This section summarizes the findings and conclusions of the action masking study. Two types of RL policies were trained, with different approaches for handling invalid actions: masking and penalization. One invalid action masking policy was trained. Eight invalid action penalization policies were trained with different penalty values.

Penalization policies with small penalty values did not earn increasing returns during training, indicating the the penalties were too small to train the the agents. Above a penalty threshold, penalization policies' returns increased throughout training, indicating that those agents learned. The masking policy learned slightly during training, earning increasing return early in the iterations before plateauing. The masking policy and the two penalization policies with the smallest penalty values completed training with similar mean return. Of the penalization policies that learned over training, all completed training with a lower mean

return than the masking policy and the two smallest-penalty penalization policies.

The trained policies were run in simulations with two benchmark policies to provide context for performance. The penalization policies complete episodes with, on median, equivalent mean trace of positional covariance, with respect to the targets in the catalog. The masking policy completes episodes with, on median, a much lower mean trace of positional covariance than the penalization policies, but larger than the benchmark policies.

The penalization policies allocate their actions either toward mostly invalid or null actions, depending on the penalty value. The penalization policies are sensitive to the penalty value, with small changes in penalty corresponding to large, nonlinear changes in action allocation behavior. The penalization policies that task mostly invalid actions do so because the penalty values are not large enough to discourage the behavior. The penalization policies that task mostly, or a large portion, null actions have too large of penalty values, which outweigh the reward for taking active actions.

The masking policy tasks null actions and wasted tasking opportunities at less of a rate than the random policy, but more than the greedy policy. The masking policy favors tasking targets with a low trace of positional covariance, which contributes to its overall higher mean trace of positional covariance than the random policy. As designed, the masking policy does not task invalid actions. The masking policy has a much lower wasted opportunity fraction than the penalization policies.

4.5 Future Work

Based on the results in this study, the following efforts are being pursued as further research.

First, because action masking shows a marked improvement over invalid action penalty

when applied to avoiding tasking non-visible RSOs, the same principle can be applied to enforcing active actions. A second action mask could be applied over the null action when targets are visible, further restricting the action space to a desireable subspace to explore during training. Such a secondary mask would eliminate the possibility of the policy wasting opportunities.

Second, alternative NN architectures and learning policy parameters should be explored. The parameters chosen for this study were done so heuristically and with the purpose of comparing invalid action masking to invalid action penalty, and as such were not optimized for absolute performance. A more systematic search of NN layers, size of layers, and learning policy parameters, such as learning rate, may yield improved policy performance.

Lastly, the observation features could be expanded to include other states that may give the learning agent more "insight" into the system dynamics. The observation features used in this study, consisting of the covariance diagonals and estimated visibility map, were chosen as a proof-of-concept to apply to the centrally controlled multi-sensor problem. An agent with more relevant information about the system may task targets more efficiently.

Chapter 5

Environment Encoding Study Methodology

This chapter describes the methodology of the environment encoding study, which builds on the techniques developed in Ch. 3 by expanding the observation space and incorporating a more advanced agent model. This study examines the performance of RL policies that have forecast state knowledge incorporated in the observation space.

5.1 Introduction

The goal of the SSA sensor allocation problem, as defined here, is to maximize utility at the final step of a fixed horizon. The assignment of sensor $m \in \mathcal{M}$ to target $n \in \mathcal{N}$ necessarily means that all targets other than n are not measured; this phenomenon represents an opportunity cost. The opportunity cost in tasking target n is not known by the myopic agent. Although RL agents are trained on a wide variety of states via experience replay, this is not the same as having online knowledge of future states and, therefore, RL agents can be categorized as myopic. Even policies that take into account previous states (e.g., with LSTM layers) are myopic because they do not have knowledge of future states.

There is an opportunity in the SSA domain to give RL policies future state knowledge.

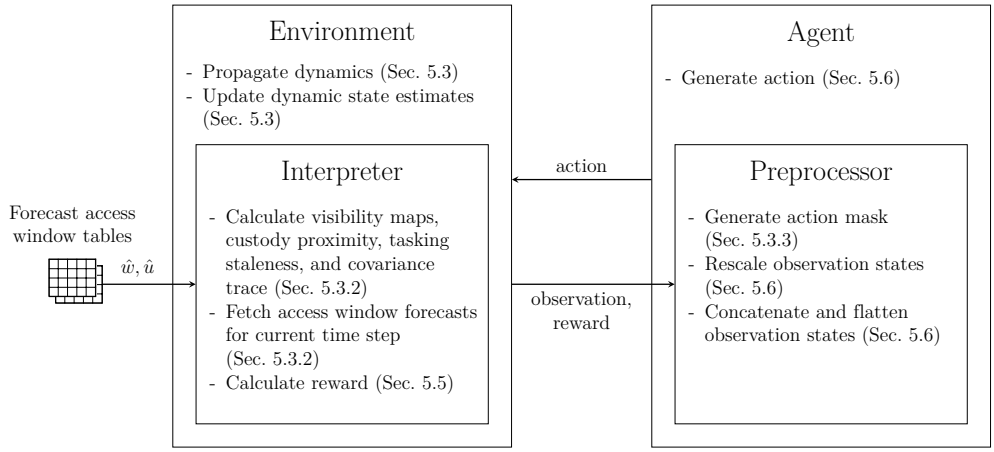


Figure 5.1: Overview of system model.

Because orbital dynamics are well known and mostly[1] deterministic, one can predict future states, within the limitations of the estimator, when given the current state of the RSO. An RL agent may better approximate the opportunity cost when it has knowledge about future states, even if that knowledge is imperfect. As a result, the agent would out-perform an agent without such knowledge, whether in terms of training or deployed metrics. This study incorporates future (predicted) knowledge into the environment in an effort to train a less-myopic RL agent.

An overview of the system model developed in this study is illustrated in Fig. 5.1.

Figure 5.1 shows the interactions between the environment (the SSA network and RSOs), and the RL agent. The environment block contains the physics and estimation functions

[1]Usually certain stochastic or partially-stochastic perturbations, such as higher-order gravitational effects and fluctuations in solar radiation pressure, are neglected.

of the model. A subprocess of the environment is the interpreter, which translates the environment's dynamic state estimates into observation states for the agent. The agent block accepts observations and reward, and outputs actions. The preprocessor, which is part of the agent block, generates the action mask from the observation and pre-treats the observation to be input to the NN. Each element in Fig. 5.1 is described in the following sections.

5.2 Custody

Several sections in this study use the concept of "custody", a term which is not universally defined in the SSA community. This section formally defines "custody", for use in this study.

Custody is qualitatively defined as the sensor network having accurate and precise enough knowledge of the position and velocity of a target to observe it at will. For an SSA sensor network, custody is a function of the knowledge of the dynamics of the RSOs, which corresponds to the ability to predict future states, and the capabilities of the sensors controlled by the network. Having very accurate dynamics models is not useful if the sensors cannot observe targets, and having exquisite sensors is not useful if the motion of the targets cannot be predicted. Accurate dynamics knowledge is assumed in this study because the scope is limited to the control of sensors, rather than the fidelity of the estimates.

Given that sufficient dynamics knowledge is assumed, custody comes down to sensor capabilities. Roughly, a sensor's ability to observe a target is a function of the target's positional covariance and the sensor's probability of detecting the target if an observation is attempted. If the positional covariance represents such a large volume of space that a sensor cannot observe a target with some minimum probability, the target is not in custody. Note that this abstracts away much of the complexity of the observation process, which could include fac-

tors such as scanning patterns, signal to noise ratio (SNR), and multiple detections, but is sufficient for the goals of this study. A probability of detection threshold can be assigned to a sensor, based on factors such as FOV and phenomenology (e.g., radar or EO), such that potential observations with an expected probability of detection less than the threshold are not tasked, as they are determined to be wasted effort[2]. If no sensor in a network has a sufficient expected probability of detection of a target, the target is said to be out of custody.

In this study, the quantitative definition of custody is to compare the positional covariance matrix (the "size" of the uncertainty) to a threshold which represents the probability of detection. This approach allows for the abstraction of sensor phenomenology and lower level sensor models. The probability of detection threshold is directly associated with a trace of positional covariance threshold ρ_m, for all sensors $m = [1, 2, \ldots, M]$. Assuming identical sensors, one covariance threshold ρ value can be used for the entire sensor network. For this study, custody of all targets in a catalog c is quantitatively defined as

$$c_n = \begin{cases} 1, & \text{tr}(P^n_{1:3}) < \rho \\ 0, & \text{otherwise} \end{cases} \tag{5.1a}$$

$$c = [c_1, \ldots, c_N] \tag{5.1b}$$

[2]Say a satellite is estimated with 99% probability to be within a 1km sphere above Australia. There may be a, say, 0.9% probability that the satellite is over the Pacific, or a 0.00001% probability that it is over Arizona. A sensor operator in Arizona is unlikely to task their sensor to the satellite given the low probability of detection, when there are other satellites the sensor could observe with higher likelihood of success.

where ρ is the custody threshold, and $P^n_{1:3}$ is the positional covariance of target n. The value used for ρ in this study is set heuristically, and is the same in all episodes.

5.3 Environment

This section describes the SSA environment model used in the environment encoding study. The environment, implemented in Gymnasium, models an arbitrary number of agents, categorized as either sensor or target, and is able to task sensors to targets on a discrete time basis [29, 59]. Agents can be terrestrially fixed or in orbit. Orbital agents' motion is modelled via two-body dynamics. Sensor measurements are the full dynamic state (position and velocity) of the RSOs corrupted by Gaussian noise. Raw measurements are not necessarily available to the RL agent, depending on observation space configuration. In this study, sensors are assumed to be terrestrial and targets are assumed to be in orbit. The terms "target" and "RSO" are used interchangeably.

The remainder of this section describes the action space, observation space, and action mask of the environment.

5.3.1 Action Space

The action space (see Eq. (3.2)) is a multidiscrete array of length M, valued $\{0, ..., N\}$, where 0 indicates the null action [29]. The mathematical definition of the environment's action space is repeated here for convenience.

$$a = [a_1, \ldots, a_M], a_m \in \{0, \ldots, N\} \qquad \text{(3.2, revisited)}$$

Table 5.1: Environment observation space for environment encoding study.

State	Dimensions
Action mask	$(N+1)M$
Continuous visibility map	NM
Time derivative of continuous visibility map	NM
Number of windows left	N
Time to next window	N
Custody proximity	N
Time derivative of custody proximity	N
Tasking staleness	N
Trace of covariance	N
Time derivative of trace of covariance	N
Total number of features	$7N + 3NM + M$

Values of a_m may be disallowed by the action mask, the details of which are described in Sec. 5.3.3.

5.3.2 Observation Space

This study modifies and expands the observation space in Ch. 3. The core environment updates the dynamics states and estimates of the agents. The interpreter, as shown in Fig. 5.1, translates the raw states into the observation features, which are encoded into the observation space.

The observation space is itemized in Table 5.1. The observation space includes time derivatives of other states, which allows the RL agent to have direct access to rate information of the environment. The states *number of windows left* and *time to next window* are used to give the RL agent future state knowledge. The remainder of this section details the items in Table 5.1.

5.3.2.1 Action Mask

The action mask is a binary array with shape $[(N+1), M]$, where a 1 denotes an action is allowed and a 0 denotes an action is disallowed. In implementation, the action mask is flattened to an $(N+1)M$-long, single-dimensional array, which is then input to the observation space.

5.3.2.2 Continuous Visibility Map

The continuous estimated visibility map is an array with shape $[N, M]$, where a value greater than 0 denotes that the associated sensor-target pair are visible to each other. Mathematically, the continuous visibility map is

$$\hat{\Omega}_{\mathrm{c}} = \begin{bmatrix} v_{1,1} & \cdots & v_{1,M} \\ \vdots & \ddots & \cdots \\ v_{N,1} & \cdots & v_{M,M} \end{bmatrix}, v_{n,m} \in \mathbb{R} \tag{5.2}$$

where $v_{n,m}$ denotes the estimated visibility function value of target n to sensor m. The values of the continuous visibility map are calculated via the the visibility function described in [63],

$$v = \arccos\left(\frac{R_{\mathrm{E}}}{||r_1||}\right) + \arccos\left(\frac{R_{\mathrm{E}}}{||r_2||}\right) - \arccos\left(\frac{r_1 \cdot r_2}{||r_1||\,||r_2||}\right) \tag{5.3}$$

where v is the value of the visibility function, R_{E} is the radius of the celestial body (in this case, Earth), r_1 and r_2 are the position vectors of two objects, and $||r_1||$ and $||r_2||$ are the magnitudes of the position vectors. Estimated positions are used in calculating the visibility map; consequently, when estimates have large errors, the visibility map may inaccurately

represent the true visibility of sensor-target pairs (a value <0 vice a value >0). Visibility misrepresentation is more likely to happen when the true visibility is near 0 (i.e., when a target is near the horizon).

Note that the continuously-valued visibility map used in the environment encoding study is different from the binary visibility map used in the environment masking study (see Ch. 3).

There is an analytical definition of the time derivative of Eq. (5.3), but the analytically-calculated values proved to be overly noisy when used on estimated states [63]. Instead, the discrete time derivative of the visibility map is used in the observation space. In implementation, both the continuous visibility map and its derivative are flattened to NM-long, single-dimensional arrays, which are then input to the observation space.

5.3.2.3 Access Windows

The *number of access windows remaining* w for each target is defined as the number of time steps between the current time t and the horizon T in which a target is visible, given the target dynamic state $x_n(t)$. The estimated number of access windows $\hat{w}$ is used in the observation space instead of the true value w because the tasking agent does not have access to the true target state. Leveraging the previously established discrete estimated visibility map $\hat{\Omega}_{\mathrm{d}}$, repeated here for convenience,

$$\hat{\Omega}_{\mathrm{d}} = \begin{bmatrix} v_{1,1} & \cdots & v_{1,M} \\ \vdots & \ddots & \cdots \\ v_{N,1} & \cdots & v_{M,M} \end{bmatrix}, v_{n,m} \in \{0,1\} \qquad \text{(3.1 revisited)}$$

the estimated number of access windows remaining $\hat{w}$ is defined as

$$\begin{aligned} \hat{w}_n = &\quad \sum_{k=t}^{T} \sum_{m=1}^{M} \hat{\Omega}_{\mathrm{d}}^{n,m}(k) \\ \hat{w} = &\quad [\hat{w}_0, \ldots, \hat{w}_N] \end{aligned} \tag{5.4}$$

and likewise for the true number of access windows remaining w. Note that $w, \hat{w} \in \mathbb{N}_0^N$ and the elements of w monotonically decrease in time $w_n(t) \geq w_n(t-1) \, \forall \, t, n$. However, $\hat{w}$, unlike w, is dependent on an estimated state $\hat{x}$, which has time-variable error. Therefore, the estimate number of access windows remaining $\hat{w}$ is time-variant.

As discussed in Sec. 5.3.2.2, using an estimated dynamic state can lead to differences between the true and estimated visibility maps; this can in turn lead to differences between the true and estimated number of access windows remaining. With every updated state estimate, a new estimated trajectory of access windows is generated, which may add, delete, or move in time the predicted access windows. Therefore, $\hat{w}$ does not necessarily maintain the monotonicity of w.

Let $W(t)$ be the trajectory of access windows remaining from time 0 until final time T, predicted at time t,

$$W(t) = [w(0|x(t)), \ldots, w(T|x(t))], \; \forall \, t \in [0, T] \tag{5.5}$$

where $w(k|x(t))$ is the vector of number of access windows remaining at time k, conditioned on the agents' dynamic states x at time t. Assuming perfect state knowledge, $W(t)$ can be perfectly predicted from the state at any time, so a window trajectory predicted at time t will be identical to one predicted at time $t + \tau$. Given

$$w(k|x(t)) = w(k|x(t+\tau)) \tag{5.6}$$

then Eq. (5.5) can be simplified to

$$\begin{aligned} W(t) = &\quad [w(0|x(t)), \ldots, w(T|x(t))] \\ W(t) = &\quad [w(0|x(t+\tau)), \ldots, w(T|x(t+\tau))] \\ W = &\quad [w(0), \ldots, w(T)] \end{aligned} \tag{5.7}$$

where in the final line of Eq. (5.7) the time-conditional term, $|x(\cdot)$, has been dropped for notational clarity. The same simplification cannot be made for the estimated window trajectory $\hat{W}$ because a forecast visibility window may be eliminated by an updated state estimate; conversely, a new window may be forecast with an updated estimate. Rewriting Eq. (5.5) for the estimated access window trajectory,

$$\hat{W}(t) = [\hat{w}(0|\hat{x}(t)), \ldots, \hat{w}(T|\hat{x}(t))], \; \forall \, t \in [0, T] \tag{5.8}$$

where the only difference between Eqs. (5.5) and (5.8) are that state estimates $\hat{x}$ and number of access windows remaining estimates $\hat{w}$ replace x and w, respectively. The state estimates taken at some time before a measurement update $\hat{x}(t)$ and some time after a measurement update $\hat{x}(t+\tau)$ do not necessarily yield the same number of access windows remaining, meaning that $\hat{w}(0|\hat{x}(t))$ and $\hat{w}(0|\hat{x}(t+\tau))$ are not necessarily identical. The simplification made in Eq. (5.7) cannot be made for the estimated case.

The distinction between estimated and true access window forecasts is important because it governs the information available in the observation space of the environment. It would be unrealistic for an RL agent to have access to the true forecast of number of access windows

remaining W[3]. As with using the estimated visibility map $\hat{\Omega}$ in the observation space (see Sec. 3.2), the estimated number of access windows remaining $\hat{w}$ is used here. However, continually updating the forecast for number of access windows remaining is computationally expensive; at every step the environment must propagate the state estimates of all targets from the current step t to the horizon T (in addition to propagating the true states by one step).

As an alternative to propagating a new forecast every time step, an open loop method is used to generate access window forecasts while being more computationally efficient. An estimated trajectory of access windows remaining $\hat{W}(t=0)$ is generated prior to the episode starting, as detailed in Algorithm 1.

In Algorithm 1, Θ is a utility variable used to store a time history of target visibility,

$$\Theta = \begin{bmatrix} \Theta_{11} & \cdots & \Theta_{1T} \\ \vdots & \ddots & \vdots \\ \Theta_{N1} & \cdots & \Theta_{NT} \end{bmatrix}, \Theta_{n,k} \in \{0,1\} \tag{5.9}$$

where $\Theta_{n,k} = 1$ indicates target n is visible to the sensor network at time k.

The estimated trajectory of access windows remaining $\hat{W}$ is stored as a lookup table. On every step, the environment returns the next element $\hat{w}(k)$ in the lookup table as an observation feature. The open loop method, while less computationally expensive, sacrifices the accuracy of the non-monotonic $\hat{W}$. A discussion of this trade is worth considering.

[3]If a forecast is perfectly true, the term "forecast" loses the implied uncertainty, and is less a forecast and more a prophecy.

Algorithm 1 Forecast number of access windows remaining

procedure CALCNUMWINDOWS($\hat{x}(0), N, T, \Delta t$)
  $\hat{x} \leftarrow \hat{x}(0)$
  $\hat{\Theta} \leftarrow$ Allocate with shape $[N, T]$
  for $k = 0 : T$ **do**
    $\alpha \leftarrow$ Allocate with length N
    $\hat{\Omega}_{\mathrm{d}} = \mathrm{CalcVisibilityMap}(\hat{x})$
    for $n = 0 : N$ **do** ▷ Iterate through all targets
      $\hat{\Omega}^n_{\mathrm{d}} = \hat{\Omega}_{\mathrm{d}}[n, :]$
      $\alpha[n] = \mathrm{Is1In}(\hat{\Omega}^n_{\mathrm{d}})$ ▷ Return 1 if target is visible to any sensor, 0 otherwise.
    end for
    $\Theta[:, k] = \alpha$
    $\hat{x} = \mathrm{Propagate}(\hat{x}, \Delta t)$
  end for
  $\Theta' = \mathrm{FlipColumns}(\Theta)$ ▷ Reverse order of columns
  $\Theta'_{\mathrm{cum}} = \mathrm{Cumsum}(\Theta')$ ▷ Cumulative sum along columns
  $\hat{W} \leftarrow \mathrm{FlipColumns}(\Theta'_{\mathrm{cum}})$
  return $\hat{W}$
end procedure

The open loop fetching of access windows remaining can generate values which conflict with the estimated visibility map $\hat{\Omega}_{\mathrm{c}}$, which is also in the observation space. For example, at a given step, the number of windows remaining for a target could be zero, $w_n(k) = 0$, yet the visibility map could show, at that instant, that the target is visible to a sensor ($\Omega^{n,m}_{\mathrm{c}} > 0$). In this example, the visibility map is correct because it is operating on the latest available state estimate, whereas the estimated number of windows remaining was set at the beginning of the episode. In practice, instances such as this occur infrequently, and the open-loop method generally provides estimated access window data close to the more accurate, continuously-updated method. If state estimates are of reasonable quality, there is little practical difference between the two methods. This sort of discrepancy in observation space is unlikely to have significant impact on training the RL agent because the magnitudes of the discrepancies are small and infrequent.

The *estimated time remaining until the next access window* $\hat{u}$ is included in the observation

space, and is defined as

$$\begin{aligned} \hat{u}_n &= \hat{t}_w^n - t \\ \hat{u} &= [\hat{u}_0, \ldots, \hat{u}_N] \end{aligned} \tag{5.10}$$

where $\hat{u}_n$ is the time remaining until the next access window for target n and $\hat{t}_w^n$ is the estimated time of the next access window for target n. The time until the next access window is calculated via Algorithm 2, which uses the utility variable Θ defined in Eq. (5.9). Similar to calculating the estimated number of access windows remaining $\hat{W}$, Algorithm 2 forecasts $\hat{u}$ for the entire episode a priori, resulting in the lookup table

$$\hat{U}(0) = \left[\hat{u}\left(0|\hat{x}(0)\right), \ldots, \hat{u}\left(T||\hat{x}(0)\right)\right] \tag{5.11}$$

where $\hat{u}\left(t|\hat{x}(0)\right)$ is the estimated time remaining until the next access window, given the current time t and initial state estimate $\hat{x}(0)$.

When there are no access windows remaining in an episode, the time until the next access window is set to a large real number. The same discrepancies exist between the open-loop and continuously updated time until next access window that are present as with the number of access windows remaining.

The time derivative of $\hat{u}_n$ is

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\hat{u}_n &= \frac{\mathrm{d}}{\mathrm{d}t}\hat{t}_w^n - \frac{\mathrm{d}}{\mathrm{d}t}t \\ &= 0 - 1 \\ &= -1. \end{aligned} \tag{5.12}$$

The estimated time until the next access window $\hat{u}$ decreases monotonically in time and with a constant slope. The time derivative of $\hat{u}$ is not included in the observation space because it does not contribute new information to the RL agent.

Algorithm 2 Forecast time until next access window

```
procedure CalcTimeToWindows(Θ, Δt)
    N, T ← shape of Θ
    Û ← allocate with shape [N, T]
    count ← ∞                          ▷ Initialize count to infinity or some large number
    for n, θ in Θ do                                          ▷ Iterate through rows of Θ
        û ← allocate with length T
        for k = T − 1 : 0, with step − 1 do                   ▷ Iterated backwards through θ
            if θ[k] == 1 then
                count ← 0
            else
                count = count + 1
            end if
            û[k] = count
        end for
        Û[n, :] = û
        Û = ΔtÛ                                               ▷ Convert number of steps to time
    end for
    return Û
end procedure
```

5.3.2.4 Custody Proximity

The custody vector c, see Eq. (5.1), is suboptimal as an observation state because it is binary-valued, and rapidly-changing states are difficult for RL agents to learn. Instead, a smoothly-evolving surrogate for custody, called *custody proximity* c_p is used in the observation space. Custody proximity is qualitatively described as how close a target is to being out of custody, with a high value being "more" in custody, and 0 being out of custody. Quantitatively, custody proximity is defined as

$$c_p = \text{clip}(\rho - \text{tr}(P_{1:3}), 0, \rho) \tag{5.13}$$

where the clip function is defined in Eq. (2.9) and target indices have been omitted for notational clarity.

$$\text{clip}(x, a, b) = \begin{cases} a, & x < a \\ b, & x > b \\ x, & \text{otherwise} \end{cases} \tag{2.9, revisited}$$

Custody proximity smoothly varies from 0 to ρ, but is not C^1 smooth because its derivative is discontinuous when $\rho - \text{tr}(P_{1:3})$ transits 0. The discontinuity is not expected to severely impact RL training because it occurs infrequently, and can only occur once per target because custody, once lost, cannot be regained.

The discrete time derivative of custody proximity is included in the observation space. The discrete time derivative smooths over the discontinuity when custody proximity falls below 0, thereby assigning a real value when analytically the derivative would be undefined.

5.3.2.5 Tasking Staleness

Tasking staleness s is defined as the difference between the current time step t and the time step at which the target was last observed t_z^n. Tasking staleness is described mathematically as

$$\begin{aligned} s_n &= t - t_z^n \\ s &= [s_0, \ldots, s_N] \end{aligned} \tag{5.14}$$

where t_z^n is the time at which the most recent measurement of target n was taken. Tasking staleness s_n increases monotonically with time until the target n is observed, at which point $s_n = t - t_z^n = t - t = 0$. If multiple successive observations are made, s_n will remain 0 until the target is not observed. The time derivative of s is not included in the observation space because, similarly to the time until the next access window $\hat{u}$, is a constant. It does not contribute new information to the RL agent.

5.3.2.6 Trace of Covariance

The trace of the covariance matrix of each target, along with its discrete time derivative, is included in the observation space.

5.3.3 Action Mask

The environment layers three component masks to create an overall action mask. The lower level data used to create the action masks, as well as the full mask, is in the observation space so that the RL agent can use the data in training. The three action mask component layers are: non-visibility Φ_{nv}, wasted action Φ_{wa}, and custody Φ_{cu}. The overall action mask is defined by

$$\Phi = \Phi_{\text{nv}} \odot \Phi_{\text{wa}} \odot \Phi_{\text{cu}} \tag{5.15}$$

where $\odot$ is the Hadamard product (element-wise multiplication). The action mask (and all component layers) are $N + 1 \times M$ matrices where the 1st row corresponds to the null action. The non-visibility mask Φ_{nv} prevents the agent from tasking sensor-target pairs that are estimated to be non-visible to each other. The non-visibility mask is defined as

$$\Phi_{\text{nv}} = \begin{bmatrix} 1 & \cdots & 1 \\ & \hat{\Omega}_{\text{d}} & \end{bmatrix}. \tag{5.16}$$

The wasted action mask Φ_{wa} prevents the agent from tasking a sensor to null action while there is a target available to that sensor. The wasted action mask is defined as

$$\Phi_{\text{wa}} = \begin{bmatrix} p_{1,1}(\hat{\Omega}_{\text{d}}^{*,1}) & \cdots & p_{1,M}(\hat{\Omega}_{\text{d}}^{*,M}) \\ 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}, \; p_{1,m}(\hat{\Omega}_{\text{d}}^{*,m}) = \begin{cases} 0, & 1 \in \hat{\Omega}_{\text{d}}^{*,m} \\ 1, & \text{otherwise} \end{cases} \tag{5.17}$$

where the superscript $*, m$ indicates the mth column of the matrix.

The custody mask Φ_{cu} prevents the agent from tasking any sensor to a target which is out of custody. The custody mask is defined as

$$\Phi_{\text{cu}} = \begin{bmatrix} 1 & \cdots & 1 \\ c^T & \cdots & c^T \end{bmatrix} \tag{5.18}$$

where c is defined in Eq. (5.1). The custody mask, combined with the dynamics of the RSO tracking problem, add an important dynamic to the environment used in this study. Because out-of-custody targets are masked from being measured, and the status of custody is dependent on the targets' state covariances, the custody state can change only once, and only from 1 to 0. Once a target leaves custody, it can never be regained. This custody dynamic is not strictly realistic because in an operational scenario target custody could be regained by either intentional searches or serendipitous observations (both of which require measurement association and identification). However, the scope of this work is restricted

to the sensor allocation mission, so the slight compromise in realism is reasonable.

5.4 Benchmark Policies

Two benchmark policies are used to compare performance against the trained RL policies: random and greedy. Both benchmark policies are myopic; they do not have knowledge, estimated or otherwise, about future states. This section defines the benchmark policies. Examples of the benchmark policy behaviors from simulations are presented in Sec. 6.3.1.

5.4.1 Random

The random policy assigns uniformly random targets to sensors, subject to the action mask detailed in Sec. 5.3.3. Mathematically, the random policy is described as

$$\begin{aligned} a_{\text{random}} &= [a_1, \ldots, a_M] \\ a_m &\sim U(\mathcal{N}_\Phi^m) \end{aligned} \tag{5.19}$$

where $\mathcal{N}_\Phi^m$ is the set of targets accessible to sensor m subject to action mask Φ.

5.4.2 Greedy

The greedy policy tasks each sensor to look at the RSO with the largest trace of positional covariance, subject to visibility constraints. If no targets are visible, the null action is chosen. Mathematically, the greedy policy is

$$
\begin{aligned}
a_{\text{greedy}} &= [a_1, \ldots, a_M] \\
a_m &= \arg\max_x \left(\varepsilon_{\text{na}}, \text{tr}(P^1_{1:3})\hat{\Omega}^{1,m}_{\text{d}}, \ldots, \text{tr}(P^N_{1:3})\hat{\Omega}^{N,m}_{\text{d}}\right)
\end{aligned}
\tag{5.20}
$$

where $0 < \varepsilon_{\text{na}} \ll 1$ denotes the value of the null action. In the event where there are no RSOs are visible to a sensor, the value of all active actions is 0. The null action subsidy ε_{na} is necessary to prevent a value tie, as ties are resolved by a uniform random selection among tied values, which could violate the mask. Only the greedy policy uses a null action subsidy.

5.5 Reward Function

Three component reward functions are combined to create the overall reward function used in this study. Each component reward function is designed to promote complimentary behaviors in the RL agent. The component reward functions are: Statistic of population uncertainty, custody, and information gain. The components and their corresponding equation numbers are summarized in Table 5.2.

Table 5.2: Reward function component equations.

Component Reward Function	**Equation Reference**
Statistic of population uncertainty	Eq. (5.21)
Custody	Eq. (5.22)
Information Gain	Eq. (5.23)

The remainder of this section describes the component reward functions and the combination method. For notational clarity, all equations omit time dependency.

5.5.1 Statistic of Population Uncertainty

The *statistic of population uncertainty component reward function* assigns reward based on the median of trace of covariances matrices of the target catalog. The intent of this statistical measure-based reward function is to incentivize the RL agent to manage the uncertainty of the entire RSO catalog. By accounting for RSOs that are not currently accessible by any sensors, the agent may be encouraged to task targets when they *are* accessible, anticipating that their uncertainty will later degrade while out of view.

The statistic reward function is defined as

$$r_1 = c\sigma\big(\text{med}(\{\text{tr}(P^1_{1:3}), \ldots, \text{tr}(P^N_{1:3})\})\big) \tag{5.21}$$

where σ is the sigmoid function as defined in Eq. (3.6), reprinted here for convenience,

$$\sigma(x) = \quad \frac{1}{1+\exp(-k(x-x_0))} \tag{3.6, revisited}$$

and c is a scaling factor discussed in Sec. 5.6. A negative value is used for the steepness parameter k such that larger reward is given for smaller median of covariance, thereby incentivizing lower catalog uncertainty. The median is used instead of the mean to mitigate the impact of outliers, but any scalar statistical measure of catalog uncertainty would meet the intent of the component reward function. The parameters used in Eq. (5.21) are in Table 5.3.

The statistic of population uncertainty reward function is smooth and in this study decreases monotonically (although this need not be the case if a large sensor network with global coverage were used). This component reward function has a relatively small magnitude of response

Table 5.3: Parameters used in population statistic component reward function, Eq. (5.21).

Parameter	Value
x_0	1×10^4
k	-1×10^{-4}

to actions by the agent because decreasing any single target's covariance (through tasking) does not significantly impact the median of the entire catalog. This light responsiveness makes the statistic component reward function a good candidate for augmentation with a more dynamic component reward function.

5.5.2 Custody

The *custody component reward function* is the most direct in terms of rewarding policies that are fit-for-use (keeping targets in custody). The objective of this reward function is to incentivize maintaining custody of targets, while simultaneously not tasking actions on targets whose covariance values are well-below the custody threshold. Reward is equal to the fraction of targets that are in custody,

$$r_2 = \frac{1}{N} \sum_{n=1}^{N} c_n \tag{5.22}$$

where c_n is defined in Eq. (5.1). Equation (5.22) decreases monotonically in time because custody, once lost, cannot be regained.

This component reward function exhibits step decreases, the magnitudes of which are larger when a small RSO catalog is used because the loss of a single target represents a larger fraction of the total number of targets. The custody component reward function is somewhat

sparse; it only changes value when a target's trace of covariance transits a threshold. The custody component becomes less sparse with larger RSO catalogs, as targets more frequently drop out from custody relative to a small catalog. RL agents can have difficulty learning with sparse rewards because the differentiating variable (reward) does not provide as much information about the relationship between observed states and actions compared to a non-sparse reward. There can be many time steps between when a target is accessible by a sensor and when that target is lost, making correlation by the RL challenging. The sparseness and time separation behaviors of the custody component reward function makes augmentation with other component reward functions important.

5.5.3 Information Gain

The *information gain component reward function* is conceptually a higher-variance, more sparse alternative to the *statistic of population uncertainty component reward function* (see Sec. 5.5.1). The reward per step is the sum of information gained between the predicted (a priori) covariance values P^- and the estimated (updated) covariance values P for all targets,

$$r_3 = \frac{1}{2}\sum_{n=1}^{N} \log\left(\frac{|P_n^-|}{|P_n|}\right). \tag{5.23}$$

Because $P^- = P$ for targets that are not measured, those targets do not contribute to the reward. For targets that are measured, $|P| \leq |P^-|$, so their contributing reward is always greater than 0. When no targets are tasked, reward is 0; this happens more often in scenarios with few RSOs or poorly-located sensors. When targets with very low covariance are tasked, reward is near 0; this happens most often when a sensor is tasked to the same target multiple times in a pass. The first tasking results in high information gain, but the subsequent taskings in the pass have low information gain.

The information gain component reward function is responsive to agent actions; reward is immediately given when a target is tasked, making correlation easier for the RL agent. The information gain component is more sparse than the other reward function components (Eqs. (5.21) and (5.22)), but also more dynamic. Information gain tends to occur in bursts of a single, high magnitude value, followed by a few (depending on time step) small values, then a period of 0 reward. Scenarios with larger RSO catalogs that present many access windows to the sensor network are more likely to exhibit more dynamic information gain behavior. The sparsity of the information gain component reward function makes it a good candidate for augmentation with a reward function that behaves more steadily in time.

5.5.4 Combined Reward Function

The objective of combining the component reward functions is to create a reward function that incentivizes the behaviors driving the individual components, while buttressing their weaknesses. The component reward functions are each z-score normalized, then summed to create the overall reward function [64]. Z-score normalization is used to transform the mean of the component reward functions to 0 and the standard deviation (STD) to 1 so that the magnitude of the reward function is not dominated by a single component. The reward function is

$$r = \sum_{i=1}^{3} \frac{r_i - m_i}{s_i} \tag{5.24}$$

where r_i is defined in Eqs. (5.21) to (5.23) and m_i and s_i are the respective mean and STD of samples of the ith component reward function. The sample sets used to calculate m_i and s_i are gathered from representative simulations prior to training.

Table 5.4: Environment configuration used for reward coefficient generation, training, and validation.

Parameter	**Coefficient Generation**	**Training**	**Validation**
Time step	100 sec	100 sec	100 sec
Num. sensors M	1	1	1
Sensor location	(0°N, 0°E)	(0°N, 0°E)	(0°N, 0°E)
Custody Threshold ρ	4000	4000	4000
Num. RSOs N	100	{2 to 10 (step 1), 15 to 40 (step 5)}	{2 to 10 (step 1), 15 to 40 (step 5)}
Horizon T	200	{100, 200}	200
Num. Episodes N_{ep}	2	Varies (Sec. 5.7)	50

The return is defined as the cumulative reward received at the end of an episode,

$$R = \sum_{t=0}^{T} r(t) \tag{5.25}$$

where $r(t)$ is the reward received at time step t.

5.5.5 Coefficient Generation

To generate reward function coefficients for Eq. (5.24), two single-episode simulations are run with different benchmark policies (greedy and random). The parameters of the simulations are in Tables 5.4 and 5.5.

One episode is run with each benchmark policy (greedy and random) on an environment with one sensor and 100 targets uniformly distributed within a narrow inclination band. The component reward functions, Eqs. (5.21) to (5.23), are calculated at each time step. The coefficients m_i and s_i for the component reward functions are calculated by taking the mean and STD, respectively, of the combined data sets from both episodes. The coefficients

Table 5.5: Resident space objects initial conditions for reward coefficient generation, training, and validation (SMA: semi-major axis, RAAN: right-ascension of ascending node, AL: argument of latitude).

Orbital Parameter	**Coefficient Generation**	**Training and Validation**
SMA	$U(R_\mathrm{E}+300, R_\mathrm{E}+800)$ km	$R_\mathrm{E}+400$ km
eccentricity	0	0
inclination	$U(-\pi/8, \pi/8)$ rad	0 rad
RAAN	$U(0, 2\pi)$ rad	0 rad
AL	$U(0, 2\pi)$ rad	0 rad

Table 5.6: Reward function coefficients for Eq. (5.24).

$\mathbf{i}$	$\mathbf{m_i}$	$\mathbf{s_i}$
1	-0.59	1.11
2	0.31	0.15
3	4.45	5.13

for the component reward functions are shown in Table 5.6. These coefficients are used in the training and Monte Carlo analysis (Secs. 5.7 and 5.8, respectively). Reward function coefficients were also calculated with other catalog sizes with similar results, which are shown in appendix C.2. The coefficients are insensitive to changes in catalog size.

5.6 Neural Network Architecture

The NN architecture used in this study consists of a FC network, followed by a single LSTM layer, and then by an action mask. Raw observations $\mathcal{O}$ are passed through a preprocessor before being input to the FC network. The architecture is illustrated in Fig. 5.2.

The preprocessor assembles the action mask from the component mask data (see Sec. 5.3.3), and then appends the final action mask to the observations while, in parallel, feeding it to

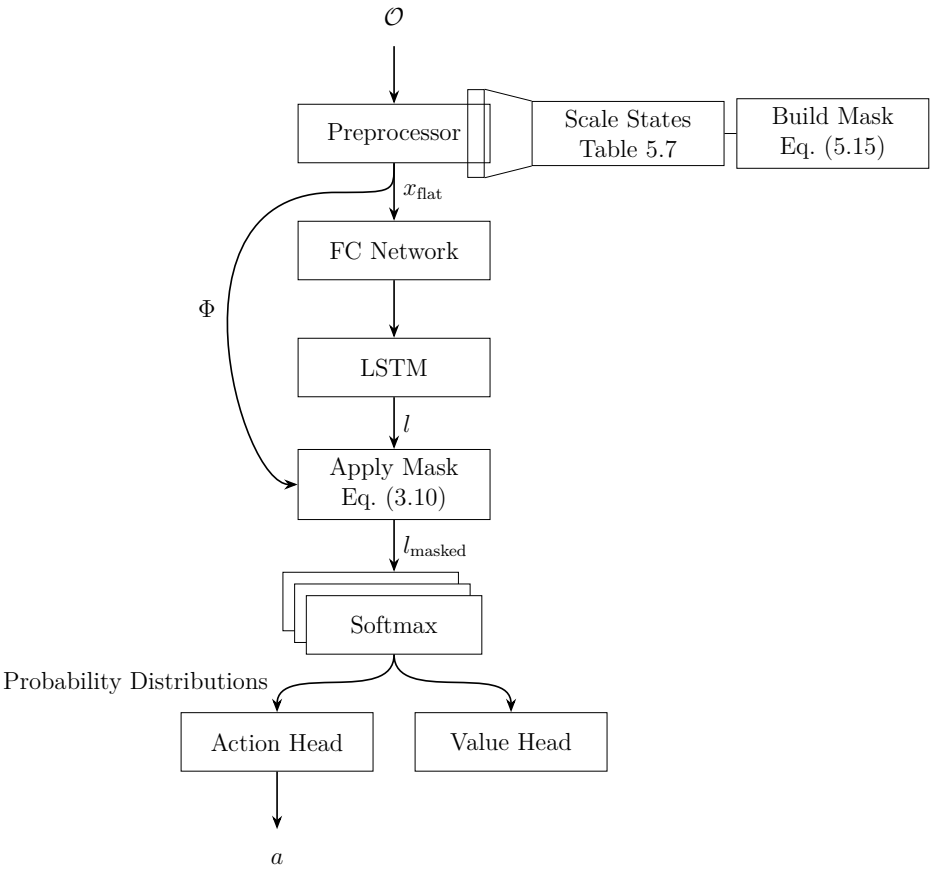


Figure 5.2: Neural network architecture used in environment encoding study.

Table 5.7: Scaling parameters used in preprocessor.

Observation State	Scaling Factor
Action mask	1
Continuous visibility map	1
Time derivative of continuous visibility map	1×10^{-3}
Number of windows left	1×10^{-1}
Time to next window	1×10^{-3}
Custody proximity	1×10^{-3}
Time derivative of custody proximity	1
Tasking staleness	1×10^{-3}
Trace of covariance	1×10^{-3}
Time derivative of trace of covariance	1×10^{-1}

the masking block. Many of the observation states (e.g., trace of covariance, time to next window) have generally large magnitudes, while others (e.g., action mask, continuous visibility map) have values near 0 to 1. To make the observation space more suitable for training, each observation state is linearly scaled to be near 0 to 1. Feature scaling is a common technique used in ML to improve training efficacy and efficiency [60]. Representative simulations were run using benchmark policies to empirically determine the scaling parameters. The observation state scaling parameters are shown in Table 5.7.

The preprocessor flattens the scaled observations into a 1D array to interface with the FC network. The FC network's number and size of layers is configurable. The LSTM layer size is also configurable. The logits l vector is $(N+1)M$-long, where each set of $(N+1)$ elements corresponds to a different sensor. The action mask is applied over the LSTM logits via the method described in Sec. 3.5. Action masking does not change the length of the logits vector, just the values.

The softmax blocks in Fig. 5.2 partition the logits vector into M sets of $(N+1)$ elements, then apply the softmax function to each set. The softmax function converts a set of arbitrarily-

valued elements into a set that sums to 1. The output from the softmax blocks is a set of M discrete probability distributions. The action branch accepts the probability distributions and generates the action vector a via an action selection method. During training, actions are stochastically sampled from the probability distributions to explore the action space. Because the masked actions have effectively zero probability, they are never chosen during training. During validation, exploration is not desired, so the action associated with the highest probability, per sensor, is chosen. The value head estimates the expected value of the state-action pair, and is used to update the weights of the model during training.

5.7 Training

To evaluate an RL policy, it is important to understand the context in which the policy was trained. The SSA sensor allocation problem, if not structured carefully, can be posed in such a way that training is impossible. This can happen if the performance ceiling of the scenario, however defined, is too low.

For example, if a sensor is never presented with multiple targets to choose from, there is no strategy for the RL agent to learn. In a less extreme scenario where a sensor has only a few non-trivial choices to make, the proportion of useful steps to train on is so low that the training process requires an unreasonably long time, effectively preventing the agent from learning. In a trivial scenario, where the policy has few or no choices to make, all policies will perform equivalently, making it impossible to gauge the efficacy of the trained policy. Conversely, if too many non-trivial tasking opportunities are presented to the sensor, the scenario may be too difficult for the RL agent to learn effectively. The scenario must be structured such that the RL agent has sufficiently many non-trivial choices to make. This ensures the performance ceiling is high enough to allow for successful training and

differentiation between policies during validation testing, yet not so many as to make the environment too difficult to learn.

A non-trivial opportunity is defined as a sensor having more than one action from which to choose. Because the wasted action mask (see Sec. 5.3.3) prevents the agent from choosing the null action when one or more targets are visible, the definition can be narrowed to a sensor having more than one target to choose from. The chief drivers of non-trivial opportunities in an episode are the number of sensors, the number of targets in the catalog, and the geometric relationship between sensor locations and target orbits, which determines the number and timing of access windows.

For this study, the complexity of the sensor-target geometric relationship is reduced by using a single sensor and co-located targets. Multiple RL agents are trained, each on a different catalog size, with identical dynamic initial conditions for the sensor and targets. The only difference between the targets is the initial covariance matrices, which have their diagonal elements randomly distributed (off-diagonals are initialized to zero).

Consider the impact on the observation space given the co-located targets scenario. Despite all targets having identical dynamic states, the observed states still differ between targets. The observation features that are forecast at the beginning of an episode (e.g., number of access windows remaining, time until next access window) are fixed and nearly identical; slight variations due to different initial covariances perturb the forecast states away form each other, but only slightly, assuming reasonably small initial covariances. The states that are updated in real time are initially identical for all targets, but diverge as some targets are observed by sensors, which alters their measurement covariances, and others are not. Depending on the catalog size and the capacity of the sensor network, the target states will diverge into two cohorts: those that are observed and those that are not. Those that are not measured will have identical dynamic states and therefore observation states. Those targets

that are observed by the sensor network will diverge from the unmeasured cohort and each other. The longer an episode runs, the more the observation states of the targets will diverge from each other.

Another driver of non-trivial opportunities is the episode horizon. The horizon alters the difficulty of the scenario, with a longer horizon offering more tasking opportunities, trivial or otherwise, than a shorter horizon. Policies tend to perform similarly at the beginning of an episode, when states are similar. As the actions of the policy affect future states of the environment, performance diverges throughout an episode. Episodes with longer horizons have more opportunities for the policy to affect future states, and can therefore better illuminate differences between policies. Larger horizons are also more difficult for RL policies to learn. For each catalog size, two policies are trained: one each with a shorter and longer horizon.

Yet another driver of non-trivial opportunities is the selection of custody threshold value ρ and UKF process noise Q, measurement noise R, and initial covariance P_0. Because custody, as defined in this study, is a function of trace of covariance (see Eq. (5.1)), any factors that influence the evolution of the covariance matrix also influence the custody state. Holding Q, R, and P_0 constant, decreasing ρ drives more targets out of custody, which in turn decreases the number of non-trivial opportunities. Alternatively, a too-large value of ρ results in all targets being in custody, making the state irrelevant to the tasking policy. A balance must be struck in setting the parameters of the UKF and the custody threshold such that targets are lost as an episode progresses, but not too quickly. The environment parameters used in training are shown in Tables 5.4 and 5.5. The parameters used in the UKF for this study are shown in Table 5.8.

Within each catalog size, a hyperparameter search is run to examine sensitivities of the RL agent and find the best-performing policy. The hyperparameter search is performed via a

Table 5.8: Unscented Kalman filter parameters used in environment encoding study.

Parameter	Value
Process noise Q	$(1 \times 10^{-3})I_6[1, 1, 1, 0.01, 0.01, 0.01]^T$
Measurement noise R	$0.1I_6[1, 1, 1, 0.1, 0.1, 0.1]^T$
Initial covariance P_0	$U(1, 10)I_6[1, 1, 1, 0.01, 0.01, 0.01]^T$

Table 5.9: Hyperparameter grid search configurations.

Parameter	Value(s)
Learning rate	$1 \times 10^{-5}, 1 \times 10^{-6}$
Activation function	tanh, ReLU
FC net configuration	[50], [100],[250], [250, 50], [100, 50], [250, 100]
LSTM layer size	32, 64

grid search strategy, in which all combinations from given sets of values are trained. The hyperparameter search parameters are shown in Table 5.9. In the "FC net configuration" row of Table 5.9, the number of values inside the brackets indicate the number of layers; the values themselves indicate the size of layers.

The configuration of the PPO algorithm is shown in Table 5.10. The number of episodes N_{ep} per iteration varies by episode horizon, and can change if many episodes in an iteration are truncated. The number of episodes per iteration is generally around 20 for $T = 100$ and 40 for $T = 200$. PPO is used as the training algorithm.

5.8 Validation

A Monte Carlo simulation is run with selected RL agents and the benchmark policies. The best-performing trained policy per catalog size is selected for the Monte Carlo simulation. Each selected RL policy is validated against the benchmark policies (greedy and random).

Table 5.10: Training parameters used in environment encoding study.

Parameter	**Value**
Clip parameter ϵ	0.3
Discount factor γ	0.9
Learning rate	Varies by trial
Training Batch Size	4000
SGD Minibatch Size	128
Training Iterations	20
Exploration Method	Stochastic Sampling

Each catalog size is run for $N_{\text{ep}} = 50$ episodes. All episodes were run using the parameters in Tables 5.4, 5.5 and 5.8; random parameters are independent, identically, and uniformly distributed. Monte Carlo runs were performed for the longer horizon case ($T = 200$), as the longer horizon better-illustrates differences between policies (Ch. 6 elaborates on this point).

The number of targets at which the performance floor and ceiling diverge can be identified by evaluating the benchmark policies' performance on the range of scenarios; any catalog size less than the divergence threshold will not show a meaningful difference between random and greedy policy performance. The RL should be unable to learn on scenarios with catalog sizes less than the divergence threshold. If the training scheme and NN architecture are able to learn the scenario, the trained policy should learn on scenarios with a catalog size greater than the divergence threshold. The lowest level of RL policy performance should be equivalent to that of the random policy because the NN weights are initialized randomly. Meaningful learning will have occurred if a trained policy outperforms the random policy on scenarios with a catalog size greater than the divergence threshold.

5.9 Metrics

This study uses several metrics to evaluate the performance of tasking policies. This section introduces the metrics which have not previously been defined.

Custody fraction C, the proportion of targets in custody, is useful for comparing simulations with different numbers of targets. It is defined as

$$C = \frac{1}{N} \sum_{n=1}^{N} c_n \tag{5.26}$$

where c_n is the binary custody of a single target and is defined in Eq. (5.1). Custody fraction is bounded on [0, 1].

The standard metric for evaluating training of reinforcement learning agent is mean episode return, defined as the mean of the cumulative reward received over all episodes at the final iteration of training. In this study, a supplementary metric is required for analyses that compare policies trained with different horizons because the bias of the mean episode return correlates with horizon. A longer episode will by definition earn more cumulative reward than a shorter episode. Therefore, the mean custody fraction $\bar{C}(t)$ is also used as a metric, formally defined as,

$$\bar{C}(t) = \frac{1}{N_{\text{ep}}} \sum_{k=1}^{N_{\text{ep}}} \big(\frac{1}{N} \sum_{n=1}^{N} c_n(t) \big) \tag{5.27}$$

where N_{ep} is the number of episodes over which the mean is being taken and $c_n(t)$ is the custody status of target n at at time t. In the context of training, N_{ep} is the number of episodes in a training iteration; in the context of a Monte Carlo simulation, N_{ep} is the number of episodes in the simulation. This document makes the context clear every time N_{ep} is used.

The mean custody fraction at the end of an episode is denoted as $\bar{C}(T)$.

Mean return $\bar{R}$ is the average return R among all episodes of a training iteration. Mean return is mathematically defined as

$$\bar{R} = \frac{1}{N_{\text{ep}}} \sum_{k=1}^{N_{\text{ep}}} R_k \tag{5.28}$$

where R_k is the return of a single episode, as defined in Eq. (5.25). Delta-mean return $\Delta\bar{R}$ is also used as a metric to evaluate the trained policies; it is the difference in mean return $\bar{R}$ between the final and initial iterations of training, defined as

$$\Delta\bar{R} = \bar{R}_{N_{\text{ep}}} - \bar{R}_0 \tag{5.29}$$

where $\bar{R}_0$ and $\bar{R}_{N_{\text{ep}}}$ denote the mean return from the initial and final training iterations, respectively. Delta-mean return $\Delta\bar{R}$ is a measure of how well the agent learned relative to its initial state. An agent with high $\Delta\bar{R}$ can underperform an agent with low $\Delta\bar{R}$ if the later has higher mean return at the end of training $\bar{R}_{N_{\text{ep}}}$, so $\Delta\bar{R}$ is best-used as a compliment to $\bar{R}_{N\text{ep}}$.

Because the error in the estimated state of the target can cause the visibility map (Eq. (5.2)) to incorrectly identify a target as visible, failed observations can occur. A failed observation occurs when a sensor is tasked to a sensor that is not visible. The failed observation fraction F_{fo} is the number of failed observations in an episode divided by the number of attempted observations. Null actions are not included in attempted observations. The failed observation fraction F_{fo} is defined as

$$F_{\text{fo}} = \frac{\sum_{k=0}^{T}\sum_{m=1}^{M} x_{\text{fail}}}{\sum_{k=0}^{T}\sum_{m=1}^{M} x_{\text{attempt}}} \tag{5.30a}$$

$$x_{\text{fail}} = \begin{cases} 0, & a_m = 0 \\ 1, & \Omega_{\text{c}}^{a_m,m} \leq 0 \\ 0, & \text{otherwise} \end{cases} \tag{5.30b}$$

$$x_{\text{attempt}} = \begin{cases} 0, & a_m = 0 \\ 1, & \text{otherwise} \end{cases} \tag{5.30c}$$

where Ω_{c} is the *true* visibility map. Note that the indices of Ω_{c} are defined on $[1 \leq n \leq N, 1 \leq m \leq M]$, so when evaluating Eq. (5.30b), the action a_m is first checked for a 0 value (the null action). If a_m is 0, then x_{fail} is 0 and the visibility map Ω_{c} is not evaluated, avoiding an out-of-bounds error. The failed observation fraction F_{fo} is not defined when the number of attempted observations in an episode is 0; this does not happen in practice.

Depending on the environment initial conditions, number of targets, and the aggressiveness of the custody action mask definition, a sensor may have few or many non-trivial tasking opportunities during an episode; see Sec. 5.7 for discussion. The primary use of the non-trivial opportunities metric in this study is in evaluating the training process. For RL policies, the fraction of time steps in which a sensor has a non-trivial opportunity F_{nt} can be used to quantify how much useful training data the agent has access to (a trivial opportunity is not useful for learning). Secondarily, the non-trivial opportunity fraction F_{nt} can also be a complimentary measure to custody fraction C. If two policies have equivalent performance in terms of custody fraction C, the policy that has a greater non-trivial opportunity fraction F_{nt} would be preferable for a sensor network because it allows more freedom of action for an

individual sensor to schedule observations within a tasking window. In this way, non-trivial opportunities are an indicator of performance capacity, although not necessarily performance, of a policy. The non-trivial opportunity fraction is defined as

$$F_{\text{nt}} = \frac{1}{TM} \sum_{k=0}^{T} \sum_{m=1}^{M} x_m, \quad x_m = \begin{cases} 1, & \sum \Phi^{*,m} > 1 \\ 0, & \text{otherwise} \end{cases} \tag{5.31}$$

where $\Phi^{*,m}$ denotes the m'th column of the action mask matrix (see Eq. (5.15)), and the time dependency of Φ has been omitted for clarity [4]. The term $\sum \Phi^{*,m}$ is the sum of the elements in the m'th column of the action mask matrix. If there is more than one action available to the mth sensor, meaning more than a single one in a column of Φ, the opportunity is non-trivial. The non-trivial opportunity fraction F_{nt} is time- and sensor-averaged, and is bounded on $[0, 1]$. For this study, there is only a single sensor, so Eq. (5.31) simplifies to

$$F_{\text{nt}} = \frac{1}{T} \sum_{k=0}^{T} x_k, \quad x_k = \begin{cases} 1, & \sum \Phi > 1 \\ 0, & \text{otherwise} \end{cases}$$

where Φ in this case is a 1d array.

The mean trace of positional covariance $\bar{P}_{1:3}$ is taken with respect to all targets in a single episode. The definition is revisited here,

[4] This definition of non-trivial opportunity fraction should not be conflated with the inverse of the number non-trivial opportunities N_{nt} defined in Eq. (3.16a). Each equation relies on a slightly different definition of the action mask Φ. In Eq. (3.16a), wasted opportunities are possible, whereas in Eq. (5.31), and indeed all of Chs. 5 and 6, wasted opportunities are masked out.

$$\bar{P}_{1:3}(t) = \sum_{n=1}^{N} \mathrm{tr}(P^n_{1:3}(t)) \qquad \text{(3.4, revisited)}$$

where N is the number of targets and t is the time step. The trace function is omitted from $\bar{P}_{1:3}$ for notational brevity. The mean of mean trace of positional covariance $\bar{\bar{P}}_{1:3}$is taken with respect to a group of episodes, defined as

$$\bar{\bar{P}}_{1:3}(t) = \frac{1}{N_{\text{ep}}} \sum_{k=1}^{N_{\text{ep}}} \bar{P}^k_{1:3}(t) \qquad (5.32)$$

where N_{ep} is the number of episodes being averaged and $\bar{P}^k_{1:3}(t)$ is the mean trace of covariance of episode k at time t.

Because custody C, once lost, cannot be regained, it is a monotonically decreasing function of time. It is useful to know the time, if it exists, at which custody value is reached during an episode; call this a custody threshold. The time at which a history of custody values $C(t)$ reaches a threshold x is defined as

$$t_x = \arg\min_t |C(t) - x| \qquad (5.33)$$

where $x \in [0, 1]$. If the custody time series $C(t)$ does not reach the custody threshold x, then t_x is undefined.

Chapter 6

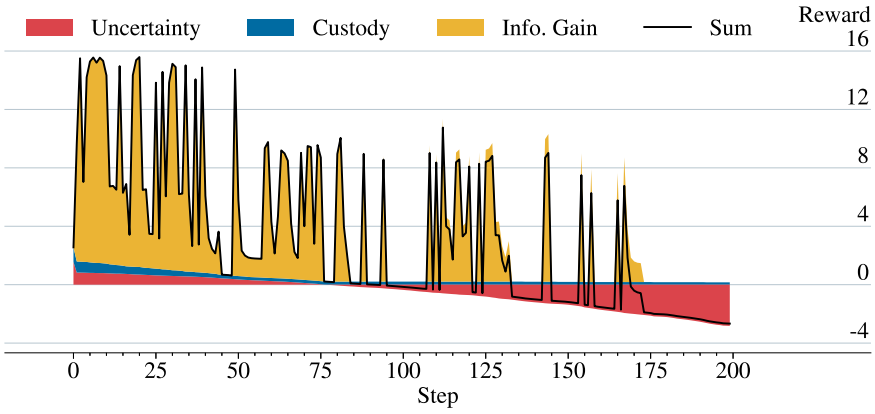
Environment Encoding Study Results

This chapter discusses the results from the environment encoding study, the methodology of which is described in Ch. 5. The results are presented and analyzed in the following sections. Section 6.1 analyzes the results from the simulations that generated the reward coefficients, which were then used to train the RL agents. Section 6.2 analyzes the training results. Section 6.3 analyzes the results from a Monte Carlo simulation of the trained and benchmark policies. Lastly, Sec. 6.4 summarizes the results and conclusions of the study.

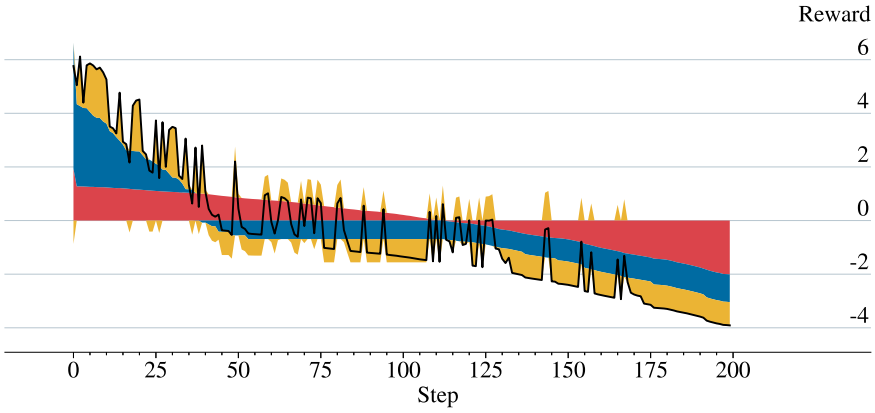
6.1 Reward Function Coefficients

Two episodes were simulated with benchmark policies to generate reward function coefficients, as shown in Eq. (5.24). This section presents and analyzes the results from these simulations, examining the reward components before and after z-score normalization.

The time histories of the component rewards and their sums for the random policy are shown in Fig. 6.1. The combined normalized reward in Fig. 6.1b shows more dynamic range than the combined raw reward in Fig. 6.1a, as designed via the normalization process. The raw combined reward (Fig. 6.1a) tends to be very low or very high, whereas the normalized reward is more evenly distributed among the range of values. The high variance behavior (the sharp increases and decreases in reward) are caused by the information gain component reward function Eq. (5.23), which is zero when a target is not tasked and near-zero on successive



(a) Raw

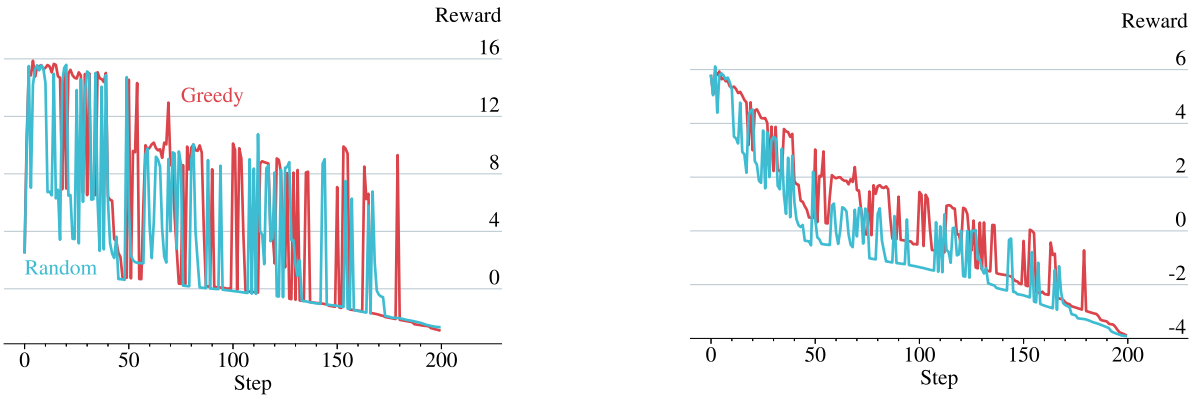


(b) Normalized

Figure 6.1: Time histories of raw and normalized reward function components (random policy).

taskings in a series. This can be seen by the large information gain spikes in Fig. 6.1a that dominate the reward signal.

The on-average downward trend in the combined reward in Fig. 6.1 is a combined contribution of the statistic of population uncertainty and custody component reward functions, which are Eqs. (5.21) and (5.22), respectively. As the episode progresses, the median trace of covariance increases, causing its component of the reward to decrease. This is correlated with the fraction of targets in custody C decreasing, decreasing its component of the reward



(a) Raw

(b) Normalized

Figure 6.2: Time histories of (a) raw and (b) normalized reward functions. Two episodes are shown, one for each benchmark policy.

function. There is an inflection point around step 40, more easily seen in Fig. 6.1b, where the slope of the reward increases (while still remaining negative). This inflection is caused by the custody fraction component, which decreases rapidly at the beginning of the episode, then to a lesser rate of decrease after about 40 steps.

The combined reward is shown for the greedy and random policies in Fig. 6.2. The greedy and random policies have similar reward time histories. However, the two policies are more distinguished from each other in Fig. 6.2b than Fig. 6.2a because normalization decreases the influence of the information gain relative to the other components, which dominates the signal in the raw data. This can also bee seen in Fig. 6.1, where the magnitude of the information gain component is much less in the normalized reward (Fig. 6.1b) than the raw reward (Fig. 6.1a). Figure 6.2b shows a generally larger reward for the greedy policy than random. This outcome matches the intuition that a heuristic policy should earn more reward than a random policy. More detailed analysis comparing the benchmark policies is presented in Sec. 6.3.

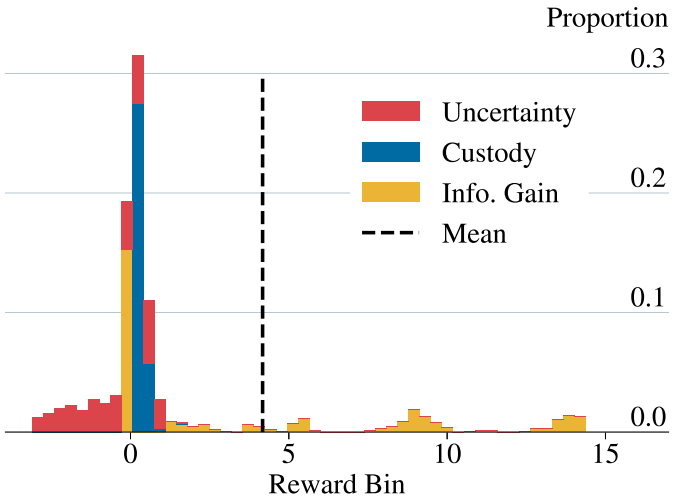
Distributions of the component rewards are shown in Fig. 6.3. The combined raw reward

has a strong mode near zero, and multiple smaller modes at larger magnitudes. The primary mode (from about -4 to 4) is left-skewed, due to contributions from the uncertainty component (median trace of covariance). The combined raw reward is not normally distributed, which can lead to sparse rewards during training. This is not conducive to training agents.

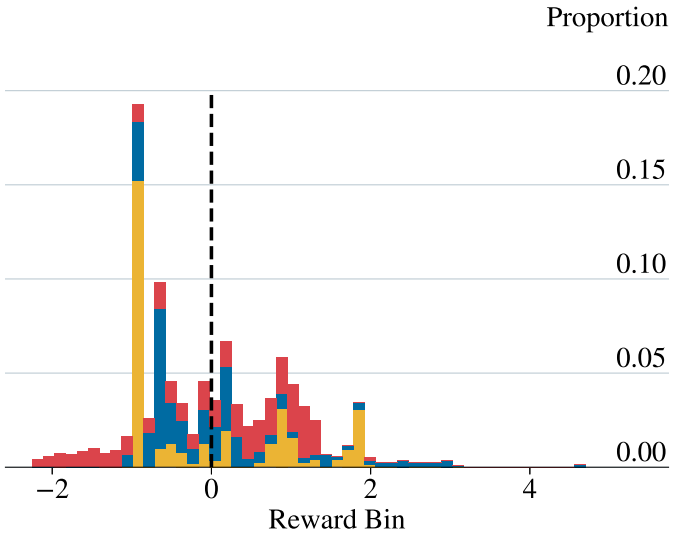
The components of the raw reward signal in Fig. 6.3a vary in their levels of concentration around different magnitudes. The custody component is the most tightly distributed of the components, with all of occurrences near zero. This is expected because the range of the custody fraction is $[0, 1]$, which is small compared to the domain in Fig. 6.3a. The uncertainty component is less concentrated, but still close to zero and slightly left-skewed. The custody and uncertainty components are unimodal. This is expected because they are both smoothly-varying values that are not subject to sudden changes.

The information gain component is the most widely distributed in Fig. 6.3a, with multiple modes and most of its occurrences concentrated near zero. The large concentration near zero is caused by successive taskings of the same target. When a target is tasked the first time in an access window, the information gain is large, meaning the covariance is reduced by a large amount. When a target is tasked soon after it was last tasked, the information gain is small because the covariance is already small.

The other modes, those away from zero, in the information gain component are caused by tasking targets with a large covariance, resulting in a large information gain. Because targets can go varying amounts of time without being tasked, and because they start the episode with uniformly distributed initial covariance values, the covariances of the targets at any given time can have a diverse range of values. The diverse range of covariances is reflected in the information gain component of the reward function, which also has a diverse range. This would seem to suggest that the distribution of the information gain component should be uniform, other than a large peak near zero. However, the distribution is not uniform; it is



(a) Raw



(b) Normalized

Figure 6.3: Proportional distributions of (a) raw and (b) normalized reward functions. Data from two episodes, one using each benchmark policy, are combined in the distributions.

multimodal. This is because of the cyclical nature of orbital dynamics and access windows. During a given time period, the sensor has access to some number of targets $n < N$. The sensor has only enough time during the period to task a fraction of the accessible targets $n_t < n$. The rest of the targets are not tasked and so their covariances continue to grow until the next access window. The cohort of n_t un-tasked targets represents a cluster of covariance values that grows at roughly the same rate. When the cohort becomes accessible to the sensor again, the sensor can task some new fraction of the cohort, thereby accumulating more information gain via the reward function, and the process continues.

Figure 6.3b shows that the normalized reward signal is more tightly distributed overall and with zero mean, as designed. The custody component is spread more uniformly throughout the range of values. The extreme values of information gain in Fig. 6.3a are brought in-family with the other reward components in Fig. 6.3b. The mean trace of covariance distribution is largely intact. The large peak near -1 in Fig. 6.3b corresponds to the peaks in custody and information gain components in Fig. 6.3a. However, the peak is less dominant relative to the overall distribution of the normalized reward, which is a positive attribute for a reward signal.

The influence of information gain, which dominates the raw reward, is lessened in the normalized reward; it is more evenly distributed with the other components. Overall, the normalized reward function is more dynamic in time and normally distributed in magnitude than any of the individual reward components or the raw combined reward, which is a desireable property for a reward function.

Table 6.1: Training performance grouped by horizon.

Metric	Horizon			
	100		200	
	Mean	STD	Mean	STD
Mean Return $\bar{R}$	506.10	14.10	878.30	81.60
Delta-Mean Return $\Delta\bar{R}$	0.50	4.20	1.40	16.00
Mean Final Custody Fraction $\bar{C}(T)$	0.86	0.12	0.68	0.12
Mean Episode Length	100.00	0.00	199.90	0.70

6.2 Training Analysis

This section presents and analyzes results from training RL agents on the SSA sensor allocation problem. The section is divided into three parts. First, the differences in training with respect to horizon (the number of steps in an episode) are analyzed. Second, training performance with different target catalog sizes is examined. Lastly, the sensitivity of the system to hyperparameter selection is presented and discussed.

6.2.1 Horizon

This section analyzes the training performance of the RL agents with respect to episode horizon. All agents were trained on two horizons, $T = 100$ and $T = 200$, to examine training under different difficulty levels (the longer horizon being the more difficult case). The parameters for training are shown in Sec. 5.7. Table 6.1 shows the performance metrics statistics for the trained agents, grouped by episode horizon. In Table 6.1, the mean of each metric is taken with respect to the final step in all episodes of the final iteration of training. The number of episodes in a training iteration is different depending on the horizon T, and varies slightly depending on how many episodes are truncated (see Sec. 5.7 for discussion).

The mean return $\bar{R}$ is greater for the longer horizon because the the agent has more steps to earn reward, as expected. Mean return varies more (has a higher STD in Table 6.1) in the longer horizon case because the target covariances have more time to diverge from one another and there are more possible action trajectories, expanding the potential reward space. Delta-mean return $\Delta\bar{R}$ is small, near zero, in both horizon cases, and buried in the variance of both itself and mean return $\bar{R}$. A positive delta-mean return $\Delta\bar{R}$ would indicate that the agents earned more average reward as the number of training iterations increased, which would be a sign of successful training. The near-zero $\Delta\bar{R}$ in Table 6.1 indicates that the agents, on average and with respect to the horizon, did not train well.

The mean final custody fraction $\bar{C}(T)$ inTable 6.1 is smaller for the large horizon case $T = 200$, an expected result due to the higher difficulty of the longer horizon. The variance in custody fraction C is driven by two factors: the actions of the agent and the natural dynamics of the system. The system dynamics contribute roughly the same amount of variation in uncertainty (estimate covariance), regardless of catalog size or horizon. The agent actions are more unpredictable, and cannot be qualitatively characterized ahead of time. The STD of the mean final custody fraction $\bar{C}(T)$ is identical for both horizon cases, indicating that the trained policies act similarly regardless of horizon. Because the contribution of the system dynamics to the mean final custody fraction variance is equivalent for both horizons, any difference in mean final custody fraction variance must be from differences in policy influence.

As a computational savings measure, episodes are truncated if all targets are lost from custody. The mean episode length in Table 6.1 is equal to the horizon in the $T = 100$ case, with zero variance, which indicates that no episodes were truncated due to all targets being lost. The mean episode length in the $T = 200$ case is slightly less than the horizon, with a non-zero variance, indicating a small number of episodes were truncated. Episodes being

truncated during training is neither unexpected nor emblematic of poor training; it is simply a signal that the agents are being stressed, which is an indicator that the environment is not *too* easy (it says nothing about the environment being too difficult). The fact that episodes were truncated during the longer horizon training, and not during the shorter horizon training, shows that the cases lie on opposite sides of a difficulty threshold; this is helpful for evaluating the agents' performance.

Taken as a whole, agents in both horizon cases did not train in terms of delta-mean reward $\Delta\bar{R}$. However, this is not the whole picture, as evaluating the agents as grouped by horizon lumps all catalog sizes together. Catalog size is a key driver in environment complexity, and grouping environments of very different complexity does not give a full assessment of performance. Catalog size is analyzed in the next section.

6.2.2 Catalog Size

As discussed in Sec. 5.9, episode return R is correlated with horizon T. This correlation makes direct comparison of R between trials with different horizons specious, and so a different metric is required to make the comparison. In this section, mean final custody fraction $\bar{C}(T)$, defined in Eq. (5.27), is used to compare the trials with different horizons.

The distributions of the mean final custody fraction $\bar{C}(T)$ for all catalog sizes are approximately Gaussian. Fig. 6.4 shows the quartiles of $\bar{C}(T)$ plotted against catalog size. Each box in Fig. 6.4 represents the 48 policies trained under the hyperparameter grid search itemized in Table 5.9. Two series are shown, one for each episode horizon.

Normally-distributed performance metrics are typical in RL training, so the general shapes of the distributions in Fig. 6.4 are not surprising. As expected, mean custody fraction is higher for the shorter horizon than the longer horizon because it becomes more difficult for

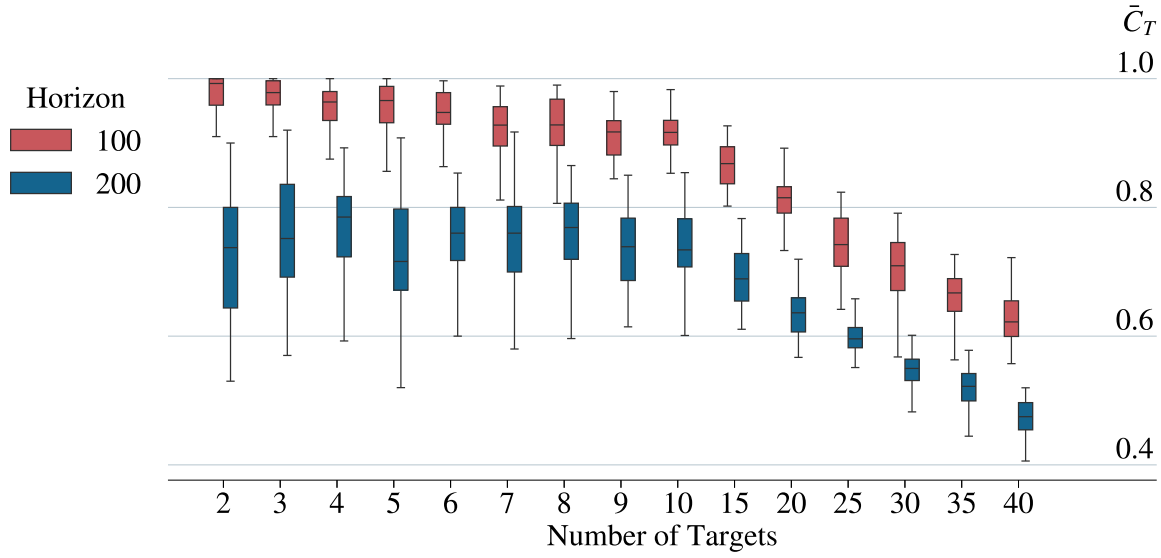


Figure 6.4: Distributions of mean final custody fraction $\bar{C}(T)$. Mean taken with respect to episodes of final training iteration. Quartiles taken with respect to all 48 trained policies per horizon-catalog-size bin.

a policy to maintain custody of targets as the episode length increases. Custody remains relatively constant for small numbers of targets, indicating that the policies are not stressed for catalogs less than around ten. Performance variance is less (narrower interquartile range) for larger catalogs. This is a property of the custody fraction metric C, see Eq. (5.27), which is more volatile for smaller catalogs. Statistics of mean final custody fraction $\bar{C}(T)$ are shown in Table 6.2.

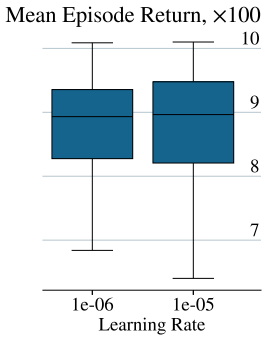
The values in Table 6.2 detail the trends illustrated in Fig. 6.4. For the short horizon, the mean of $\bar{C}(T)$ generally decreases with increasing catalog size. For the longer horizon, this same trend is seen for the larger catalogs ($N \gtrsim 10$), but not for smaller catalogs, where $\bar{C}(T)$ is roughly constant. This suggests that the policies are equally stressed for small catalogs on the long horizon, but not on the short horizon. There could be a natural horizon threshold for which the average policy becomes stressed, regardless of catalog size ($\lesssim 10$). On one

Table 6.2: Mean final custody fraction $\bar{C}(T)$ statistics grouped by horizon and catalog size.

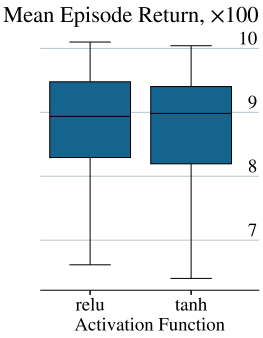
Catalog Size	**Horizon**			
	100		**200**	
	Mean	**STD**	**Mean**	**STD**
2	0.97	4.58×10^{-2}	0.72	1.06×10^{-1}
3	0.97	2.89×10^{-2}	0.75	8.04×10^{-2}
4	0.95	3.99×10^{-2}	0.77	8.40×10^{-2}
5	0.96	3.59×10^{-2}	0.73	6.87×10^{-2}
6	0.94	4.16×10^{-2}	0.75	7.89×10^{-2}
7	0.92	4.24×10^{-2}	0.75	7.12×10^{-2}
8	0.92	4.60×10^{-2}	0.76	6.97×10^{-2}
9	0.91	4.18×10^{-2}	0.73	6.56×10^{-2}
10	0.91	3.51×10^{-2}	0.74	4.65×10^{-2}
15	0.87	3.50×10^{-2}	0.69	4.21×10^{-2}
20	0.81	4.33×10^{-2}	0.63	2.73×10^{-2}
25	0.74	4.70×10^{-2}	0.60	2.85×10^{-2}
30	0.71	4.76×10^{-2}	0.55	3.54×10^{-2}
35	0.66	5.12×10^{-2}	0.52	2.96×10^{-2}
40	0.62	4.45×10^{-2}	0.47	2.96×10^{-2}

side of the horizon threshold, policy performance degrades gracefully as new targets are introduced. On the other side, policies simply fall to some level of custody fraction C_T by the end of the episode. In the case of the long horizon, that custody fraction level is about 0.75, and the horizon threshold is between 100 and 200, based on Table 6.2. The fact that custody starts to degrade for larger catalog sizes in the short horizon case shows that there is catalog size threshold, as well as a horizon threshold, for which policies become stressed, which makes intuitive sense.

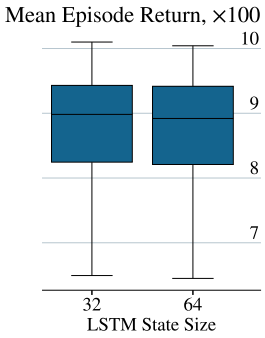
In Table 6.2, the STD of mean final custody fraction $\bar{C}(T)$ is approximately constant with respect to catalog size for the short horizon trials ($T = 100$). For the long horizon trials, the STD is roughly constant for small catalog sizes ($N \lesssim 10$) and decreases with increasing catalog size for for large catalog sizes. Because custody fraction C changes in steps, it is insensitive to small changes in gross system covariance. The short horizon leaves less time, relative to the long horizon, for covariances to grow and diverge from each other; all of the covariances stay relatively close together. This phenomenon is attenuated when examining the final mean custody fraction $\bar{C}(T)$, which further collapses differences in covariance because it ignores large (out of custody) targets. This is seen, in Table 6.2, in the relatively constant STD of $\bar{C}(T)$ for the short horizon case. Examining just the large catalogs in the long horizon case, the smooth decrease in STD of $\bar{C}(T)$, along with the mean, indicates that the trained policies gradually become *more consistent* in maintaining custody while at the same time becoming *less able* to maintain custody of targets. For the long horizons and small catalog sizes, the STD of $\bar{C}(T)$ is constant, indicating that policy consistency is insensitive to catalog size within the small regime ($N \lesssim 10$).



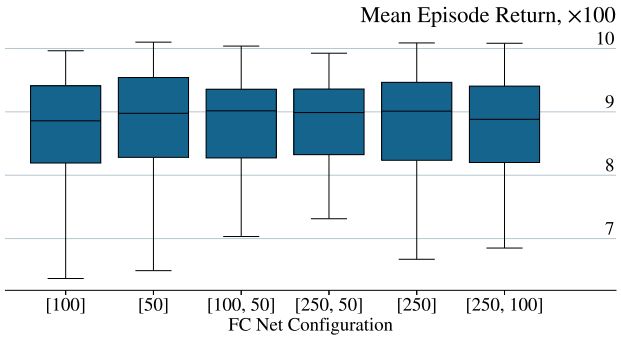
(a) Learning rate



(b) Activation function



(c) LSTM state size



(d) FC network configuration

Figure 6.5: Performance distributions for trained policies. Boxes shows quartiles ($T = 200$).

6.2.3 Hyperparameters Sensitivity

The ML training process can be sensitive to hyperparameters. A hyperparameter grid search was conducted to broadly sample different configurations and find the best-performing policies. See Sec. 5.7 for details.

The distribution of mean episode return at the final iteration of training $\bar{R}_{N_{\text{ep}}}$ is normally distributed. Box plots of the $\bar{R}_{N_{\text{ep}}}$ distribution are shown in Fig. 6.5. Trials are marked as outliers if they lie outside of 1.5 times the interquartile range from the first and third quartiles. The data from policies trained on the longer horizon ($T = 200$) are shown. The policies trained on the shorter horizon ($T = 100$) are in appendix C.3 and are similar.

Table 6.3: Mean episode return statistics varying hyperparameter configurations ($T = 200$). In Table 6.3c, column "FC Net Config." the number and value of items in each list denotes the number and size of layers, respectively.

(a)

Learning Rate	Mean	STD
1e-06	876.7	80.4
1e-05	879.9	82.8

(b)

Activation Function	Mean	STD
relu	880.2	79.8
tanh	876.4	83.5

(c)

FC Net Config.	Mean	STD
[50]	878.3	89.1
[100]	871.6	86.2
[250]	884.3	77.9
[100, 50]	880.4	77.4
[250, 100]	875.2	79.4
[250, 50]	880.1	80.2

(d)

LSTM Size	Mean	STD
32	881.7	80.4
64	874.9	82.8

For each hyperparameter examined, the distributions were similar regardless of the value selected, as shown by the nearly identical quartiles in Fig. 6.5. The statistics of the mean episode return distributions are shown in Table 6.3.

The data in Table 6.3 confirms that the training process is insensitive to hyperparameter selection. The mean of mean returns varies little, in relation to STD, with hyperparameter configuration. Also, the STD of mean returns is similar across all hyperparameter sweeps; the magnitudes of STD are similar across all sub-tables of Table 6.3. This indicates not only that the system is insensitive to particular hyperparameters, but that all hyperparameters have a roughly equal influence on the training process.

The hyperparameter configurations for the best-performing, in terms of mean return, long-horizon policies in each catalog size bin are shown in Table 6.4. The column labelled "Cohort Mean Return" corresponds to the policies with the indicated catalog size and the same horizon ($T = 200$). The best-performing policies for the short-horizon case are shown in

Table 6.4: Best-performing policy hyperparameter values for long horizon ($T = 200$).

Catalog Size	Learning Rate	Activation Function	FC Net Config.	LSTM Size	Mean Return	Cohort Mean Return
2	1×10^{-5}	ReLU	[50]	32	992	916
3	1×10^{-5}	ReLU	[250, 100]	32	1008	930
4	1×10^{-5}	tanh	[100, 50]	32	1000	942
5	1×10^{-6}	ReLU	[250]	32	1009	926
6	1×10^{-5}	tanh	[250]	64	995	925
7	1×10^{-5}	tanh	[100, 50]	32	1004	927
8	1×10^{-5}	ReLU	[50]	32	1010	930
9	1×10^{-5}	ReLU	[250, 50]	64	993	917
10	1×10^{-6}	tanh	[100]	32	996	922
15	1×10^{-5}	tanh	[250, 100]	64	984	890
20	1×10^{-5}	tanh	[250, 50]	64	951	856
25	1×10^{-5}	ReLU	[250, 50]	32	888	827
30	1×10^{-5}	tanh	[50]	64	874	782
35	1×10^{-5}	ReLU	[250, 50]	64	865	761
40	1×10^{-5}	ReLU	[100]	64	791	726

appendix C.4.

Table 6.4 shows that the higher learning rate (1×10^{-5}) is generally the best choice among the two values tried. The learning rate of 1×10^{-5} is associated with 13 of 15 of the best-performing policies. Even though there was little difference in mean return $\bar{R}$ with respect to learning rate (see Table 6.3a), the higher learning rate had a slightly higher mean and greater STD. This means that, for a given catalog size, the single trial with the highest mean return $\bar{R}$ is likely to belong to the cohort of higher learning rates.

The values of the other hyperparameters (activation function, FC network configuration, and LSTM state size) are about evenly-represented in the best-performing policies. None of these hyperparameters have the same higher-mean/higher-STD relationship as learning rate (see Tables 6.3b to 6.3d), so the policies associated with the greatest mean return $\bar{R}$ do not

correlate strongly with particular hyperparameter values.

Given the large variance in return associated with hyperparameter configurations, see Table 6.3, any strong verdict is difficult to make regarding large catalog best-policy correlation with hyperparameter configuration. Furthermore, due to the insensitivity of custody distribution in small catalog scenarios, as discussed in Sec. 6.2.2, inferring any correlation between hyperparameters and performance for small catalogs would not be well-founded. The main driver behind hyperparameter performance differentiation is random initialization of NN weights; the system is generally insensitive to hyperparameter selection.

The best-performing policies shown in Table 6.4 are used in the Monte Carlo analysis, presented in the next section.

6.3 Monte Carlo Analysis

This section presents and analyzes the results from running a selection of trained policies, as well as two benchmark policies, in a Monte Carlo simulation. The best-performing RL policies from the hyperparameter grid search, see Table 6.4, were selected for the Monte Carlo analysis. As discussed in Sec. 6.2.1, the two horizon cases, $T = 100$ and $T = 200$, showed the same trends in training, with the longer horizon trends being more exaggerated. The longer horizon drives greater differences in policy performance due to its greater difficulty. Consequently, only the longer horizon cases were selected for Monte Carlo simulation and analysis; all of the trials analyzed in this section are from the $T = 200$ horizon case. The environment configuration for the Monte Carlo simulation is summarized in Tables 5.4 and 5.5.

This section is divided as follows. Section 6.3.1 presents time series metrics from a typical

episode to illustrate features and characteristics of the system. Section 6.3.2 discusses how tasking opportunity timing evolves as estimate uncertainty increases. Section 6.3.3 analyzes the frequency of failed observations across policies and catalog sizes. Section 6.3.4 analyzes non-trivial tasking opportunities, which quantifies "performance capacity." Section 6.3.5 analyzes the performance of the policies, primarily in terms of custody.

6.3.1 Exemplar Episodes

This section presents exemplar episodes from the Monte Carlo simulation to establish context for statistical analysis in subsequent sections. First, all targets from a single episode of the trained policy simulations are shown. Second, aggregated statistics from all targets for all three policies (trained, random, greedy) are shown. Figure 6.6 shows the trace of positional covariance for an episode under a trained policy. The custody threshold ρ is indicated by a dashed line; targets with a $\text{tr}(P_{1:3})$ higher than this value are out of custody. Individual episodes figures for the random and benchmark policies, as well as for other catalog sizes, are shown in appendix C.5.

In Fig. 6.6, drops in trace of positional covariance $\text{tr}(P_{1:3})$ occur immediately following observations. Targets are not observed if the value of $\text{tr}(P_{1:3})$ is greater than the custody threshold ρ, a behavior consistent with the design of the custody mask. When an individual target is unobserved for multiple time steps, uncertainty increases. Successive observations of the same target, seen at the bottom of Fig. 6.6 near steps 50, 120, and 180, result in diminishing returns in uncertainty decrease. The mean is less sensitive than the median to individual targets being observed; the targets with high uncertainty dominate the mean because they are orders of magnitude larger than most other targets. Eventually, near the end of the episode, most targets have high uncertainty, and the median trends closer to the

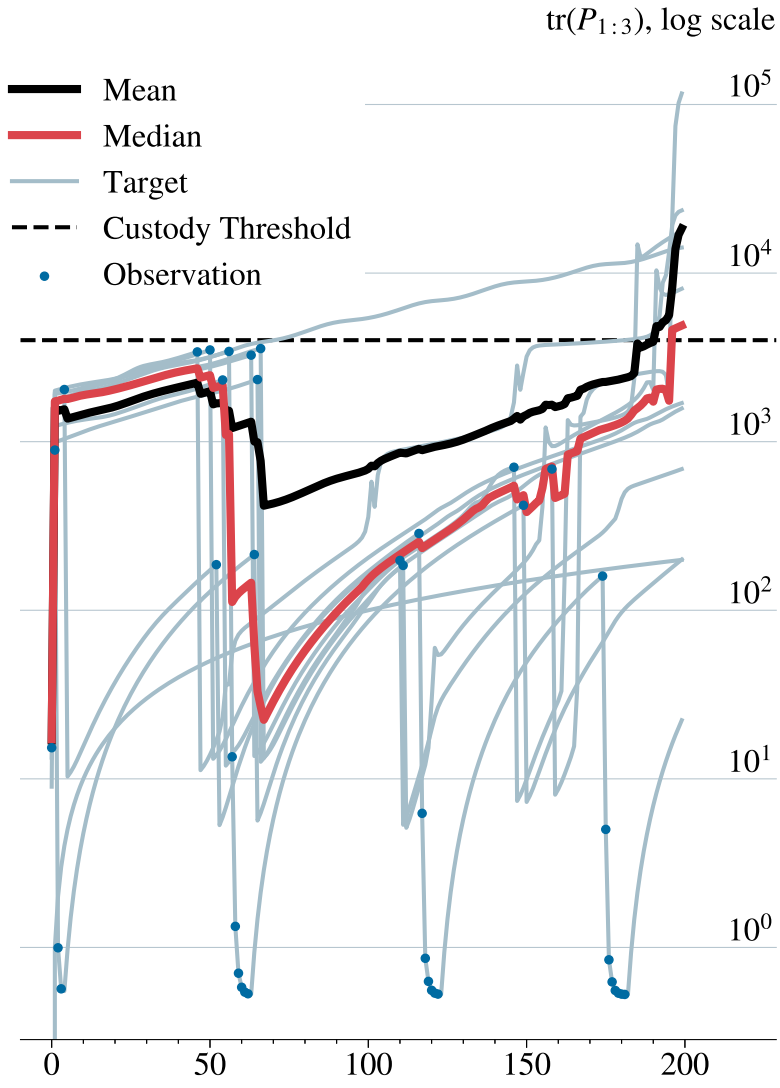
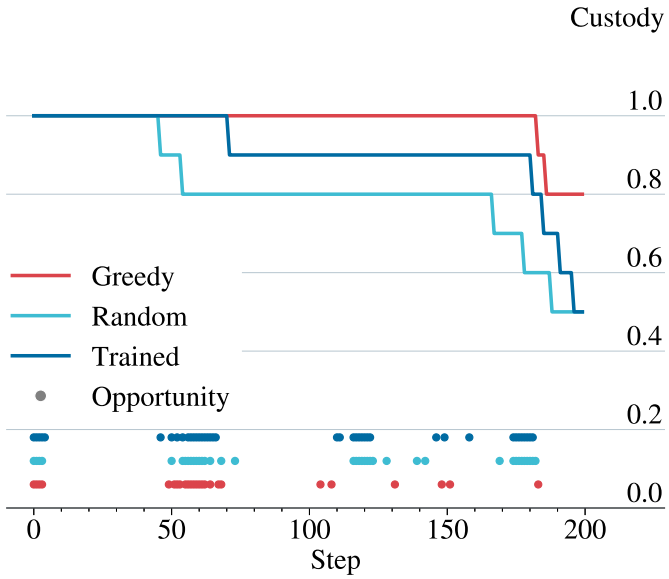


Figure 6.6: Individual target trace of positional covariance for a single episode (trained policy, 10 targets).

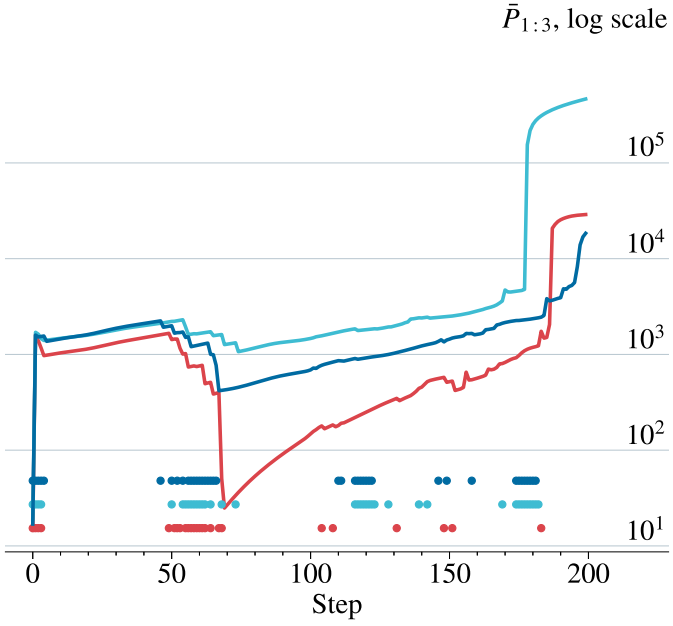
mean. When the values in the covariance matrices become very large, numerical effects in the UKF become apparent, denoted by steep decreases and increases in $\mathrm{tr}(P_{1:3})$ (seen in multiple lines around step 150).

Figure 6.7 shows the custody fraction and mean trace of positional covariance for a typical episode with a catalog size of $N = 10$. The mean for the trained policy in Fig. 6.7b is the same as that shown in Fig. 6.6. Time series data for other selected catalog sizes are shown in appendix C.5. Three policies are shown, one trained and two benchmark. Tasking opportunities are indicated with dots. As noted in Sec. 5.7, each episode has independent and identically distributed initial conditions; the data shown does *not* represent an identical scenario run multiple times under different policies. The data shown in Fig. 6.7 should not be used to make general conclusions about policy performance relative to each other, as these data represent only a single episode, and RL-derived policies have inherently stochastic behavior. This analysis is intended only to illustrate characteristics of the system, performance of policies is analyzed further in the next section.

There are several features of the time series data in Fig. 6.7 that are worth noting. In Fig. 6.7a, all policies start with full custody of the catalog ($C = 1$) and lose custody of targets later in the episode. Sometimes targets can start the episode out of custody, depending on initial covariance values, causing the initial value of C to be less than one (see Sec. 5.7 for discussion), although these instances were rare. In Fig. 6.7b, covariance immediately increases after initialization; this is caused by the the initial covariance being relatively low compared to process noise, both of which are design parameters, and is unrelated to the policies' actions. The step-downs in Fig. 6.7 correspond to targets being observed; the steps in Fig. 6.7b are more pronounced than they would be with a larger catalog size because the larger number of targets in the episode reduces the effect of any individual target being observed. Most episodes end shortly after a sharp increase in mean trace of positional



(a) Custody fraction



(b) Mean trace of positional covariance

Figure 6.7: Performance metrics and tasking opportunities for typical episode with all policies ($N = 10, T = 200$).

covariance, as shown in Fig. 6.7b; this is caused by a large enough number of target state estimates becoming poor enough that they bias the mean of the catalog to a high value. This sharp increase, when it occurs, is part of the system dynamics and reflects a stressing scenario, which is useful for evaluating policies. The goal of the scenario is not for all policies to well-maintain custody of all targets, but to stress policies enough to differentiate performance among them; environment parameters, such as horizon, were chosen to reflect this goal.

Even though both the trained and random policies in Fig. 6.7 finish the episode with the same number of targets in custody, the trained policy has a lower mean trace of positional covariance. This illustrates that although covariance and custody are related, they are only loosely coupled, particularly because a small number of poorly-observed targets can dramatically increase statistical measures of catalog covariance. This point is reinforced by the fact that the greedy policy, at the end of the episode, has the highest (good) custody fraction, but has has a slightly *higher* (worse) mean trace of positional covariance than the trained policy. This indicates that, while the greedy policy lost only a couple of targets, those lost targets were lost early and had time for their covariances to greatly diverge, increasing the mean metric to be on the same order of magnitude as that of the trained policy, which lost more targets than the greedy. This is not to say that any policy is better than the other, at this point, but to illustrate the relationship between the custody and covariance metrics.

Mean trace of positional covariance increases smoothly when there are no available tasking opportunities, which can be seen by the smoothly-increasing portions of the lines coinciding with the areas in-between the dots in Fig. 6.7b. The co-located targets in this scenario complete about 3.6 orbits in an episode, giving the sensor at most three true access windows, each that span multiple steps, to observe the targets. However, this number is corrupted by the belief state of the policy, which can increase or decreases the duration of the access

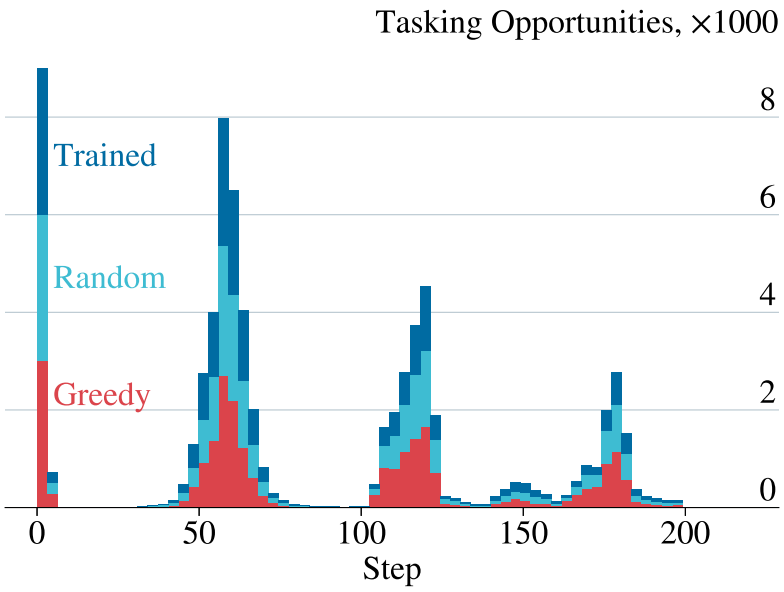


Figure 6.8: Estimated tasking opportunities for all episodes of Monte Carlo simulation.

windows, or break up a contiguous access window into multiple windows.

6.3.2 Tasking Opportunity Timing

The estimated times of tasking opportunities, which amount to the actual times of tasking opportunities because the sensor network does not have access to the true dynamic states, are shown in Fig. 6.8.

Figure 6.8 shows the full picture of the tasking opportunities shown as gray boxes in Fig. 6.7. Figure 6.7 shows a single episode from each policy, whereas Fig. 6.8 shows the estimated tasking opportunities for all episodes of the Monte Carlo simulation. In Fig. 6.8, tasking opportunities initially occur in tight clusters of time, corresponding to the initially-accurate estimated visibility (near step 0). As an episode progresses and state estimates degrade, the estimated visibility between the targets diverges. This causes the policy to have different

beliefs about which targets are visible at a given time, despite the true visibility being synchronized for all targets. The policy's divergent beliefs about visibility are reflected in the tasking opportunities, which become spread out as the episode progresses.

This divergence of tasking opportunities is shown in Fig. 6.8 by the increasingly wide modes in step ranges around 0–10, 50–75, and 100–125, and 150–190. The final double-peak in the histogram, around steps 150–190, represents a divergence in believed tasking opportunities, with the smaller peak being the result of poor estimates. Also, the overall number of tasking opportunities decreases for each mode in Fig. 6.8. This is caused by the networks losing custody of targets over time, which naturally decreases the number of tasking opportunities. Even though the simulated scenario uses co-located targets with identical true dynamic state histories, the belief states that the agents operate on diverge as episodes progress, creating a wide range of tasking behaviors in the policies.

6.3.3 Failed Observations

The fraction of failed observations per episode shows how often the estimate error is large enough such that it causes an observation to fail because the visibility status, v in Eqs. (5.2) and (5.3), is incorrectly estimated. Failed observations happen mostly when the targets are near the horizon. In these cases, a small error in the state estimate can cause the estimated visibility status to be on the opposite side of 0 from the true visibility status. The mean fraction of failed observations F_{fo}, as calculated via Eq. (5.30), is shown in Table 6.5.

The data in Table 6.5 shows that the different policies are affected similarly by failed observations. All policies have failed observations at about the same rate relative to the number of observations attempted. This is an expected behavior because the policies use the same filter design and visibility determination method.

Table 6.5: Mean failed observation fraction F_{fo} from Monte Carlo simulation ($N_{\text{ep}} = 50$).

Catalog Size	Trained	Random	Greedy
2	0.27	0.26	0.27
3	0.31	0.31	0.32
4	0.31	0.33	0.34
5	0.37	0.37	0.38
6	0.39	0.38	0.40
7	0.41	0.39	0.41
8	0.39	0.44	0.41
9	0.41	0.44	0.43
10	0.45	0.44	0.45
15	0.46	0.50	0.49
20	0.52	0.51	0.54
25	0.54	0.52	0.54
30	0.57	0.56	0.60
35	0.58	0.57	0.59
40	0.59	0.58	0.59

The fraction of failed observations F_{fo} increases with catalog size. This is a result of the increased difficulty in the scenario as the number of targets increases. As more targets are introduced, the policies are less able to maintain small errors in the state estimates. Overall larger estimate error corresponds to greater error in visibility status estimates and more failed observations.

6.3.4 Non-Trivial Opportunities

In the sensor allocation problem, as defined in this study, any tasking policy will perform similarly to another if policies are not presented with enough non-trivial opportunities (see Sec. 5.7 for discussion). Non-trivial opportunities, and the fraction of non-trivial opportunities F_{nt} defined in Eq. (5.31), are a way of measuring the potential for an RL agent to train in this environment. Policies can also affect the number of non-trivial opportunities they

are presented with by maintaining more targets in custody.

To avoid making conclusions about relative policy performance in regimes where the environment restricts performance such that all policies are expected to be equivalent, those regimes must be identified. This section analyzes the occurrence of non-trivial tasking opportunities, as defined in Eq. (5.31), in the Monte Carlo simulation to determine the threshold of catalog size above which policies are presented with enough non-trivial opportunities to differentiate performance. The divergence threshold should occur at the catalog size at which the random and greedy policies' performance becomes statistically different from each other; the performance of any policies are expected to be identical below the threshold. This threshold is useful to identify because it allows an experiment to be designed which uses the smallest possible catalog size, decreasing computation time, while still being able to differentiate policy performance.

The non-trivial opportunity fractions F_{nt} was calculated for all episodes in the Monte Carlo simulation. The distributions of F_{nt}, as grouped by policy and catalog size, have well-defined single modes, and so are visualized well as box-and-whisker plots. The distributions of non-trivial opportunity fraction F_{nt} are shown in Fig. 6.9. Each box-and-whisker diagram shows the quartiles of all episodes for a given policy and catalog size.

Overall, the median fraction of non-trivial opportunities in Fig. 6.9 is low, less than 20%, for all catalog sizes and policies. Most of the time, all targets are believed to be out-of-view of the sensor (this belief happens to be true). The low fraction of non-trivial opportunities suggest that, of all the time steps the RL policies were trained on, only a small fraction of them were useful for learning, which may have contributed to the relatively low delta-reward values discussed in Sec. 6.2. However, it is not clear what a sufficiently large enough non-trivial fraction would be to ensure the training process has capacity to enable learning. Furthermore, there are other factors that influence training that are unrelated to the environment, such as

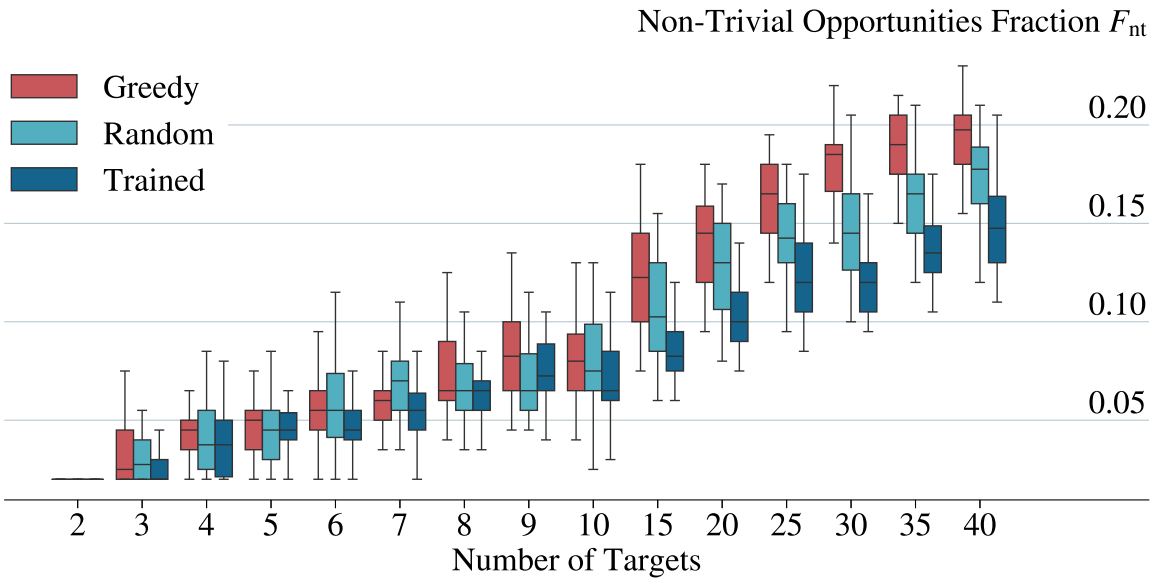


Figure 6.9: Quartiles of non-trivial tasking opportunities fraction F_{nt} for Monte Carlo simulation ($N_{\text{ep}} = 50, T = 200$).

the NN architecture and training scheme, that could also contribute to an RL agent's ability to learn. Increasing the non-trivial opportunity rate, most directly by modifying the initial conditions of the system, would, by definition, increase the difficulty of the training process, which could also prevent learning.

The median non-trivial opportunity fraction F_{nt} in Fig. 6.9 increases with catalog size, which is expected. As the number of targets increases, the agent is presented with more steps where multiple targets are available to the sensor. Similar to the trends discussed in Sec. 6.2.2, for small catalog sizes ($N \lesssim 10$) non-trivial opportunities occur at about the same median rate for all policies in a given catalog size. Below about 10 targets, policies are not easily differentiated between each other. Policies are clearly differentiated by $N = 15$, distinguishing themselves further as the catalog size increases. It is important that both the mean end-of-episode custody (Table 6.2) and the non-trivial opportunity fractions F_{nt} diverge at the same point; it shows that non-trivial opportunities are a key driver in policy performance

and confirms the initial observation in Sec. 6.2.2 that policies are not stressed for catalog sizes less than around 10. In the challenging catalog sizes ($N \gtrsim 10$), the greedy policy consistently has a greater median non-trivial opportunity fraction than the random policy, which in turn is greater than the trained policy, indicating that the trained policy has the least capacity for performance, although it is not necessarily the lowest-performer (discussed further in Sec. 6.3.5).

The random policy generally has the largest IQR of the policies, which is expected for a random action-selection process. The greedy and trained policies have similar IQR, although their medians are starkly different, particularly for large catalogs. The strong divergence between the policies in terms of median ability to generate non-trivial opportunities indicates that the policy has an influence on the tasking opportunities in an episode. Policy choice matters. However, the relatively large IQR across all policies shows that the policies' ability to generate non-trivial tasking opportunities is limited and strongly influenced by the system dynamics. The system is inherently noisy.

Although the fraction of non-trivial opportunities F_{nt} is not a direct measure of performance itself, it is a measure of the potential for performance, and the fact that the benchmark policies generate generally more non-trivial opportunities than the trained policy indicates that the trained policy has relatively less capacity for performance. Performance is directly analyzed in the next section.

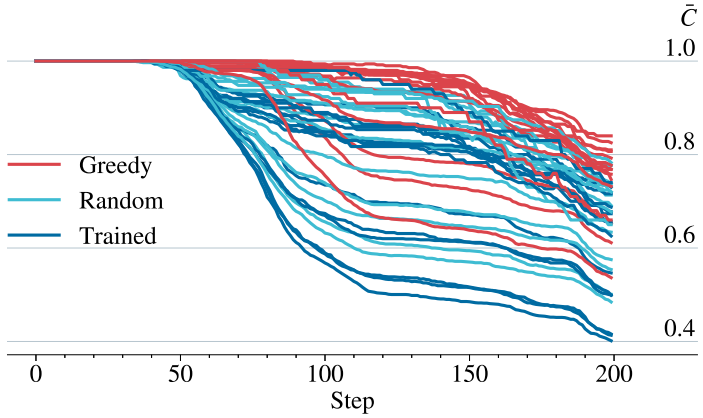
The mean and STD of the non-trivial opportunity fraction F_{nt} show the same trends as those in Fig. 6.9, and are shown in appendix C.6.

6.3.5 Custody Performance

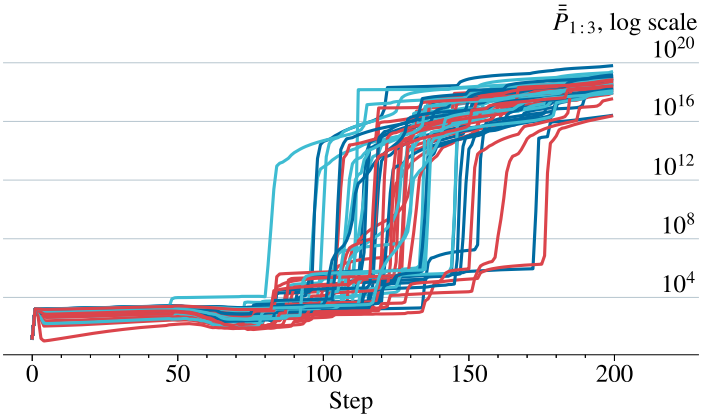
This section analyzes the performance of the trained and benchmark policies from the Monte Carlo simulation in terms of custody fraction C and positional covariance $P_{1:3}$. Figure 6.10 shows the step-wise mean of both of these metrics, plotted for all policies and catalog sizes. The points that make up each line are the mean of 50 episodes at that step. Each line represents a single policy and catalog size. Figure 6.10a shows the mean custody fraction $\bar{C}$, as defined in Eq. (5.27). Figure 6.10b plots the mean of means of trace of positional covariance $\bar{\bar{P}}_{1:3}$, as defined in Eq. (5.32). First the mean of all targets' trace of covariance is taken, then the mean is taken with respect to all episodes.

In Fig. 6.10a, the greedy policy maintains mean custody fraction $\bar{C}$ longest throughout episodes. This can be seen by the cluster of greedy lines in the upper-right of the plot, which indicates that a higher level of custody fraction C is, on average, maintained toward the end of an episode. Conversely, the random and trained policies are spread more evenly over the y-axis, with the trained policy having slightly more representation in the lower region. This lack of clustering by the random and trained policies indicates that both policies have inconsistent performance, in terms of custody fraction C, throughout an average episode.

The time at which a policy reaches a given custody threshold t_C, as defined in Eq. (5.33), is a useful metric for quantifying a policy's ability to maintain custody. Figure 6.11 shows the distribution of the number of steps t_C that a policy takes to drop to a given custody fraction C. This plot can be thought of as a view from cutting a horizontal line through Fig. 6.10a, starting at a given value on the y-axis (C axis), stopping when data points are reached, then drawing vertical lines starting at the data points, going down to the x-axis (time axis). The distributions are all unimodal, and so they are visualized with box-and-whisker plots. Each of the box-and-whiskers represents a single policy and catalog size. In Fig. 6.11, a larger



(a) Mean custody fraction $\bar{C}$



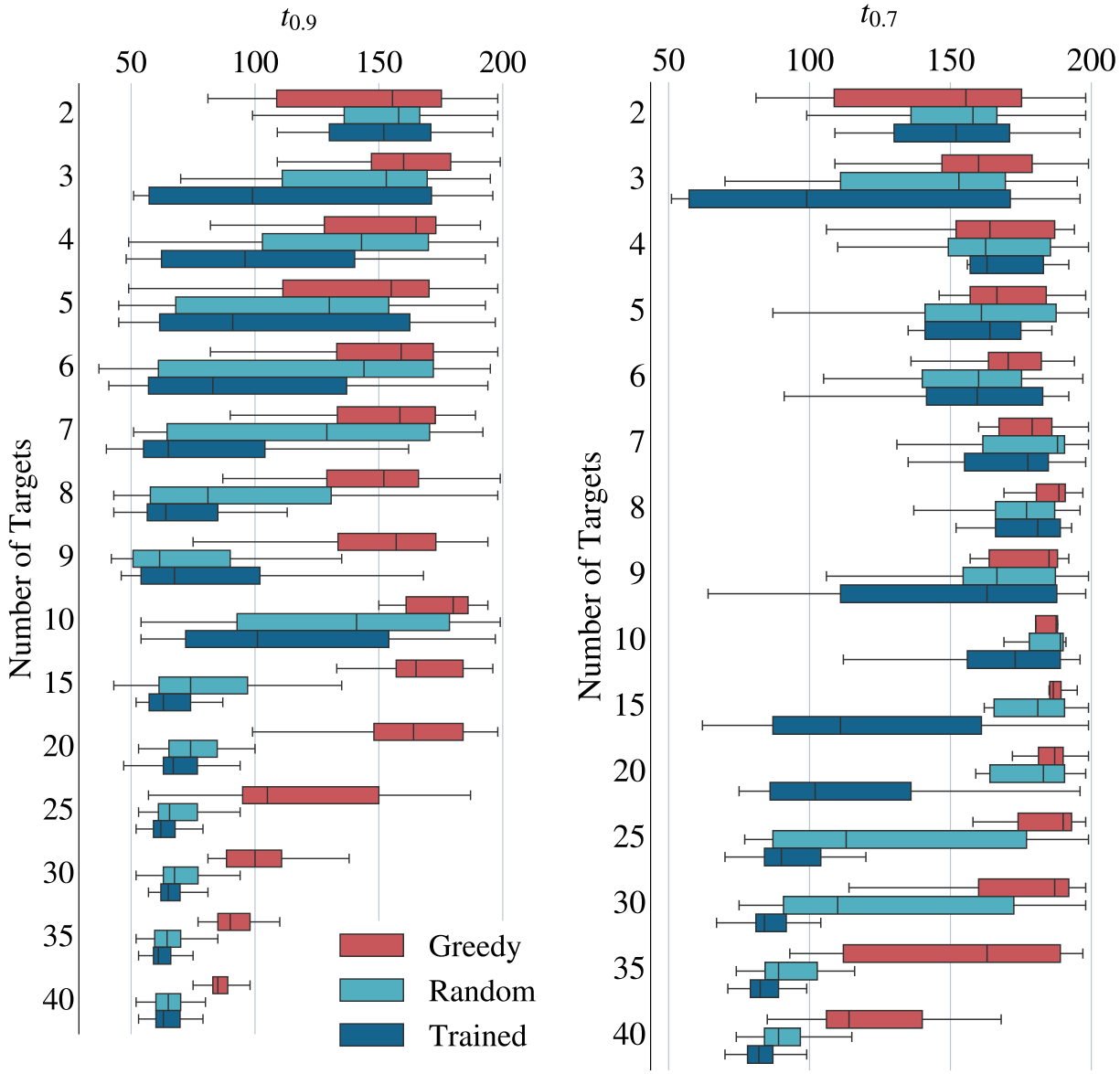
(b) Mean of mean trace of positional covariance $\bar{\bar{P}}_{1:3}$

Figure 6.10: Step-wise mean of given metrics. Each line is the mean of 50 episodes of one catalog size.

number of steps is better. In some episodes, the given custody threshold C was not reached; those episodes are omitted from the calculation and characterized by a relatively larger IQR. Such incidences are more common with the custody fraction threshold C set to lower values, in this case $C = 0.7$.

For catalog sizes of $N = 2$, all policies have equivalent performance in both C thresholds, as seen by the similar median and IQR values in Fig. 6.11. This catalog size presents too few non-trivial opportunities for the policy to make a difference. Regions of Fig. 6.11 where all policies have similar performance correspond to custody fractions C and times t in which the dynamics of the system drive the performance of the policies, not the policies themselves. In Fig. 6.11a, the greedy policy has generally the highest median number of steps to fall to the custody threshold $t_{0.9}$ for all catalog sizes, whereas the trained policies have the lowest. The IQR of the greedy policy is substantially less than the random and trained policies for most catalog sizes. Therefore, the greedy policy is more consistent in maintaining custody across episodes than the other policies. There is much less differentiation between the greedy and random policies with the lower custody threshold $C = 0.7$ (Fig. 6.11b). In a given episode the lower custody threshold $C = 0.7$ is, by definition, reached later in time (a higher t_C) than the higher threshold $C = 0.9$. In Fig. 6.11b, there is overall less policy differentiation than in Fig. 6.11a, particularly between the greedy and random policies in the small catalog sizes ($2 < N \lesssim 10$). The greedy policy maintains a higher custody fraction C than the random and trained policies for both the high custody fraction $C = 0.9$ and low custody fraction $C = 0.7$. The difference in t_C between policies is also greater (medians are further separated) for the higher custody fraction threshold $C = 0.9$ than the lower $C = 0.7$.

For small catalogs ($2 < N \lesssim 10$), the greedy policy has a consistently higher $t_{0.9}$ than the random policy. Yet, again for small catalogs, the greedy policy is about equivalent in $t_{0.7}$ to the random policy. This result shows that the greedy policy maintains custody longer than



(a) $C = 0.9$ (b) $C = 0.7$

Figure 6.11: Number of steps for a policy to fall to a given custody fraction t_C. Greater is better. Each box plot shows data from 50 episodes.

the random policy when custody values are high, early in an episode, but is about equivalent to the random policy later in the median episode. This is an expected behavior because the greedy policy is designed to be myopic in time.

Given that the greedy policy has differentiated performance from the random policy in small catalogs ($2 < N \lesssim 10$), of which there are few non-trivial opportunities and one would expect policies to perform nearly equivalently, it is reasonable to expect that a trained policies might also differentiate themselves from random in the same regime. In fact, the trained policies have generally lower median $t_{0.9}$ than the random policy for small catalogs, as seen in Fig. 6.11a. This result indicates that the trained policies are making less-random decisions than the random policy, although those decisions result in a lower $t_{0.9}$, which is undesirable behavior. However, in Fig. 6.11b, the all policies have equivalent $t_{0.7}$ (again, for small catalogs). This result indicates that at the lower custody threshold $C = 0.7$, later in the episodes, the trained policies make decisions equivalently to the benchmark policies. Therefore, at this point in time of the episodes (around step 160 to 170), and for the small catalog sizes, the custody fraction is most-influenced by the dynamics of the system, and not the policies.

For large catalogs, the trained policy has equivalent-to-fewer median steps to the custody threshold t_C than the random policy, depending on catalog size. This is true true for both custody thresholds, $C = 0.9$ and $C = 0.7$, as can be seen by comparing the trained and random policies on large catalog size, $N \gtrsim 10$, in Fig. 6.11. The trained and random policies drop to the $C = 0.9$ threshold at similar times, as seen by the similar median and IQR values for large catalogs in Fig. 6.11a. However, the trained policy has a lower $t_{0.7}$ than the random policy, as seen in the same region of Fig. 6.11b. The trained policy falls to the lower custody threshold $C = 0.7$ earlier than the random policy for large catalogs.

The trends observed in Fig. 6.11 can be summarized with respect to trends in policy, time,

Table 6.6: Statistics of $t_{0.7}$ for Monte Carlo simulation.

Catalog Size	Trained		Random		Greedy	
	Mean	STD	Mean	STD	Mean	STD
2	145	40	146	32	146	37
3	112	54	144	37	157	28
4	160	35	165	24	161	35
5	156	30	155	37	170	17
6	149	42	148	41	172	15
7	162	36	176	21	178	12
8	163	43	172	24	185	9
9	150	43	164	33	177	13
10	157	44	183	9	181	14
15	124	41	172	28	185	10
20	114	36	166	35	186	7
25	104	33	131	44	182	16
30	89	12	128	42	170	32
35	86	12	101	28	151	37
40	84	8	95	20	123	24

and catalog size. The greedy policy generally maintains a higher level of custody for longer than the other policies for all catalog sizes, although this trend is less strong for larger catalog sizes later in the episodes (see the right half of Fig. 6.11b). The trained policy maintains custody less randomly than the random policy for small catalogs early in episodes, but equivalently later in episodes, when system dynamics dominate the system. Conversely, for large catalogs, the trained policy maintains custody equivalently to the random policy early in episodes (see right half of Fig. 6.11a), but reaches the lower custody threshold $C = 0.7$ earlier than the random policy (see right half of Fig. 6.11b).

The mean and STD of $t_{0.7}$ are shown in are shown tabularly in Table 6.6. The corresponding table for $t_{0.9}$ is in appendix C.7.

The greedy policy has a generally higher mean $t_{0.7}$ than the other policies, for all catalog sizes, meaning that the greedy policy maintains a custody fraction of $C \geq 0.7$ for longer than

the other policies, on average. The greedy policy has a generally smaller STD of $t_{0.7}$ than the other policies across all catalog sizes. Therefore, the greedy policy maintains custody for longer into an episode, on average, and does so more consistently than the other policies.

For small catalog sizes, $N \lesssim 10$, the STD of the trained and random policies do not have a clear trend with respect to catalog size. In the same region, the greedy policies exhibits a downward trend in STD with respect to catalog size. For small catalogs, the random and trained policies' behavior is insensitive to catalog size, but the greedy policy is increasingly consistent (decreasing variance) with more targets.

Comparing the random and greedy policies on small catalogs, a divergence in mean $t_{0.7}$ appears at a lower catalog size than the training analysis indicates, see Sec. 6.2.2. The mean diverges at around five targets, vice ten, as seen by the sharp increase in difference in mean $t_{0.7}$ between random and greedy, and a sharp decrease in the STD of the greedy policy. This indicates that the system dynamics stop dominating the custody fraction C at around five targets, meaning policy choice matters at this threshold and greater. This divergence indicates that a trained policy may learn with catalogs greater than $N = 5$. In this case, the trained policy performance, in terms of mean $t_{0.7}$, does not clearly diverge from the random policy's until catalog sizes of around seven to ten, as shown in Table 6.6. This observation indicates that there are two catalog size thresholds relevant to the system: one at which performance differentiation is possible, as described in Sec. 6.3.4, and one at which a trained policy actually learned. The performance differentiation threshold is identified by the divergence in performance between a random policy and some other benchmark policy, in this case greedy. The learning threshold is identified by the divergence in performance between a random and trained policy. Both thresholds are not necessarily the same, and they are not the same in this study. The performance differentiation threshold occurs at around five targets, whereas the learning threshold occurs at around seven to ten.

6.4 Conclusions

This section summarizes the findings from the preceding sections in this chapter and synthesizes those findings into the major conclusions of the study.

The training process is insensitive to horizon selection. The longer horizon amplifies differences in performance among the policies that are present in the shorter horizon, but in a less pronounced way. The more-differentiated performance is caused by the filters having more time to diverge from initial conditions and each other, increasing uncertainty, and driving more targets out of custody. Training is insensitive on target catalog sizes less than about ten. Training performance degrades as catalog size increases beyond ten; this degradation is about equal for both horizon cases. The system is insensitive to hyperparameter selection. Policies are trained equivalently regardless of horizon, but the longer horizon is easier to evaluate due to the more divergent performance among policies.

The Monte Carlo analysis showed that the belief states of the policies diverge significantly, having a large influence on the actions and evolution of the system. The co-located targets scenario creates an overall low number of non-trivial tasking opportunities, which may have inhibited training progression. However, this effect is highly coupled with the training algorithm and NN architecture, and so it is unclear of the magnitude of contribution that the low rate of non-trivial tasking opportunities had on training. For challenging catalog sizes, those greater than around ten, the greedy policy generates, on average, the most non-trivial tasking opportunities, followed by the random policy, and then by the trained policies.

The custody fraction analysis shows that the greedy policy maintains a given custody fraction for longer than the other policies, on average. The trained and random policies reach given custody fractions at similar times early in episodes, on average, when custody fractions are high. When custody fractions are low, later in episodes, the random policy maintains a given

custody fraction for longer than the trained policy.

There are two thresholds relevant to the system: one at which performance differentiation is possible and one at which a trained policy has learned. In this case, the performance differentiation threshold is at around five targets, and the learning threshold is at around seven to ten targets.

The insensitivity of the training process to horizon is a positive result, as it indicates that the methodology is generalizable for different episode lengths. However, the fact that the training process is insensitive to hyperparameter selection and that overall return growth is low indicates that the RL agents did not learn on the environment. The results of the Monte Carlo simulation, in terms of a policy's ability to maintain custody of targets over time, supports this conclusion.

It is shown that there is an identifiable catalog size threshold, above which the performance of the random and greedy policies diverge. This threshold identifies the region of catalog sizes in which a trained policy can be expected to differentiate itself from a benchmark policy. This threshold is necessary to identify so that it can be determined if an RL training process is successful or not, while simultaneously allowing for the experiment design to use as small of a catalog as possible to reduce computational cost.

In the catalog sizes above the random-greedy divergence threshold, the trained policy reaches custody fraction thresholds at similar times as the random policy. The trained policies reach set custody fraction thresholds, on average, at equivalent times as the benchmark policies when presented with small catalog sizes. This result is expected given the identification of the learning and performance differentiation thresholds.

While the RL policies do not maintain custody as long as the the benchmarks, this study provides a methodology for identifying the region of catalog sizes on which a policy, RL-

based or otherwise, must be trained so that its performance can be differentiated from other policies. This study developes the non-trivial opportunities metric for evaluating the ratio of useful training steps in the training process. This study also developes a reward function which combines features of time-wise myopic information gain, whole-catalog uncertainty, and a newly-defined custody fraction to train policies which are both responsive in time and balance the uncertainty of an entire RSO catalog.

6.5 Future Work

There are three categories of future work to improve on the results of this study: the NN and training algorithm, the reward function, and the environment configuration.

The NN architecture could be modified on to improve the training process. This study examined variants of a LSTM-based network, but there are alternatives. Recent advancements in applying transformers, which have demonstrated good performance in learning sequential data, to RL could improve training performance [65]. The training algorithm could also be modified. This study examined only the "vanilla" PPO algorithm, which is the most well-represented algorithm in the SSA sensor allocation literature. However, any training algorithm which is compatible with a multidimensional action space could be used in this study. This study showed that, depending on the environment configuration, the number of steps in an episode which are trivial to the agent can be large. The training algorithm could be modified to only consider non-trivial steps during training, which would increase efficiency, and potentially performance.

This study used a single set of reward function coefficients in z-score normalization. The episodes that were used to determine the coefficients had a different catalog size and orbital configuration than the environment on which the RL agents were trained. Even though the

sensitivity of the coefficients to catalog size was determined to be low, the sensitivity to orbital configuration and very small numbers of targets was not examined. The training process could be improved by using reward coefficients tailored to the specific environment on which the RL agents are trained.

The environment configuration in this study is fairly simple, with a single sensor, co-located targets, and a maximum horizon of about three orbits. However, the environment could be made even easier for an RL agent to learn by decreasing the horizon and time step to make an episode occur over a single observation window of the sensor (about five to ten minutes). Such a configuration would present the agent with almost no trivial opportunities, making the training process more efficient.

Chapter 7

Conclusions

This dissertation has presented two studies examining environment encoding techniques for the SSA sensor allocation problem. The individual conclusions of those studies are in their respective chapters, namely Chs. 4 and 6. This chapter is divided into two sections that discuss topics that connect the two studies. Section 7.1 discusses the broad, overarching themes of the studies, and contextualizes those themes with the literature discussed in Ch. 2. Section 7.2 outlines specific lessons learned from this research which can be used to further advance the techniques developed in this dissertation.

7.1 Broad Themes

Action Space Representation

There are two primary techniques in the literature for encoding the action space of the SSA sensor allocation environment. One is for a sensor to select one of N targets, the other to select a pointing direction. The former lends itself to the tasking concept, the later to scheduling. Both techniques about equally represented in the literature, although the latest research favors the pointing direction method, as seen in Ch. 2. The research in this dissertation adds another entry to the target selection technique.

Unlike applications such as robotics and aircraft control, where the action spaces are largely

self-evident (e.g. actuate control surfaces, throttle engines), the community has not yet converged on common action space representation. This lack of a default action space representation is not surprising considering the novelty, at the time of writing, of applying RL to the SSA sensor allocation problem. The Earth-observation satellite tasking and scheduling problem, which is similar to SSA sensor management, is more mature in that there are more publications on the topic. There is an even wider variety of action space representations in the Earth-observation literature. This suggests that there are more representations to explore in the SSA sensor allocation problem. Notably, continuous action spaces are not represented in the SSA literature. The establishment of a best-practice action space representation is an open question in the field.

This research is the first to explicitly defray the tasking and scheduling problems, and focuses on the tasking portion. This distinction is inspired by the operational model of the SSN. Previous research has made no distinction between tasking and scheduling. The combined tasking-scheduling problem is hierarchical; a centralized tasking agent allocates sensors to targets, abstracting the scheduling details to the individual sensors. This subtlety in differentiating between tasking (sensor allocation) and scheduling (sensor pointing trajectories) forces the action space of a tasking agent to the form of "choose one of N." Action space representation has significant influence on the design of the observation space and RL scheme. For example, the pointing direction technique allows for a straightforward application of CNNs. The distinction between tasking and scheduling, to the extent that one is made, is a key factor when designing the action space representation because it has intrinsic implications on the observation space and RL techniques used to solve the problem.

Accounting for Invalid Actions

Regardless of how the action space is encoded, the SSA sensor management problem, to include tasking and scheduling, has invalid actions which must be accounted for in designing a solution. In the tasking problem, the invalid actions are the targets that are not accessible to the sensor(s). In the scheduling problem, the invalid actions are pointing directions which are obscured by terrain (in the case of terrestrial sensors). This research examined the former case.

In the tasking problem, the valid action sub-space is dynamic for any non-trivial RSO catalog. Targets are continually entering and exiting the FOR of a sensor. Depending on the orbital parameters of the targets, the location of the sensor(s), and the length of the episode horizon, the cycles of access windows between sensors and targets can have little regularity. This makes for a non-convex, stiff, and multidimensional surface over which to optimize. Such a problem is challenging for any algorithm to optimize, as the results of the environment encoding study show (see Ch. 6). In that study, the RL agents are not able to learn through training, even with a relatively simple scenario (co-located targets and a single sensor).

An action space representation which allows for a less dynamic, easier to optimize over, valid action sub-space may prove more practical to solve. In the representation in which a terrestrial sensor must select from a discretized set of pointing directions, the valid set, those above the horizon, is constant. Some dynamism can be incorporated into the valid action set by masking pointing directions which are known to have no targets in the corresponding FOV. Such an approach is shown in [14, 15, 51], with favorable results (see Ch. 2 for discussion). This approach likely yields a less-challenging-to-optimize reward surface because the motion of RSOs across the FOR of a sensor is smooth, making the corresponding valid action sub-space evolve more smoothly. This reward surface is likely less challenging than the

corresponding surface for the "choose one of N" action space representation.

Observation Space Representation

The observation space representation for the tasking problem, a set of tuples in which each target is a tuple of features, is inherently non-smooth with respect to the target index. The features (e.g., covariance, number of access windows remaining) in target i have no physical relation to the features in target $i+1$. This property makes the the observation-action-reward relationship nonlinear and challenging to optimize over.

Tools such as CNNs rely on integrating observation features along a dimension, usually spatial or time. If adjacent features have no physical relationship, then neither will the features of their integral. Because there is no spatial relationship between target indices, an RL agent must rely on estimating relationships within an index over time. Estimating time series relationships, particularly those with delayed or long-term impacts from actions, is more challenging, and may require more sophisticated or recently-developed tools, such as RNNs or attention networks.

7.2 Lessons Learned

Statistical Measures of Covariance are Easily Dominated

When the state of an RSO is reasonably-well estimated, the diagonals of its covariance matrix are small. In such cases, the covariance provides a useful physical sense of uncertainty; a 1-σ uncertainty ellipsoid is a good approximation of the target's state about 1-σ of the time. However, when estimates are poor and covariances are large, the Gaussian assumption

on the distribution is invalid, meaning the metric is no longer a reliable representation of physical uncertainty. Even when estimate has diverged, covariance is still useful as a metric to describe uncertainty, so long as it is not interpreted in a physical sense. This dual sense of covariance is first discussed in this dissertation in Sec. 3.9.

When quantifying a metric to represent the overall uncertainty of an RSO catalog, some statistical measure of the target covariance matrices is the most obvious choice to make, and is used in this dissertation and those works discussed in the literature review (see Sec. 2.2). Typically the covariance matrix is condensed into a scalar value by evaluating the trace, or the trace of the positional components of the matrix, giving a metric that is a interpreted as the amount of uncertainty in the dynamic state. This metric is useful because it allows the uncertainty of a target to be quantified as a single value that is grounded in estimation theory. Then a statistical (e.g., mean or maximum) measure is calculated among the population of matrix traces, giving a whole-catalog representation of uncertainty.

The problem occurs when estimates for targets diverge, driving the values in their covariance matrices to very large magnitudes very quickly. In these conditions, any statistical measure of the covariance matrices can become easily dominated by a few outliers. This domination effect is seen in Chs. 4 and 6, where the magnitudes of the "mean" covariance for some policies are orders of magnitude larger than others. The undue influence of outliers on the statistical measure can be mitigated by using a more stable measure, such as median vice mean.

If a dominated statistical measure of uncertainty is used only to evaluate policies, the weakness of the measure can be buttressed by using supplemental metrics or excluding outliers from the calculation. However, if the statistical metric is used in a feedback control scheme, such as in the reward function, the policies themselves can be affected. To illustrate, if a single target's covariance explodes early in an episode, say through an unlucky draw of

initial conditions, and the reward function is based on the mean of covariance traces, then the reward function will be saturated for the duration of the episode. Because the reward function is saturated, the RL agent is unable to gain useful action-state transitions from the episode, and the episode is not useful for training. This event, if common enough during training, can make the training process inefficient at best, and impossible at worst.

To resolve the issue of exploding covariances saturating the reward signal, the reward function can be designed via one of two methodologies. First, covariance need not be directly included in the reward function, in favor of some other, less volatile, quantification of uncertainty. This idea is explored in Sec. 5.5 with the proposal of a "custody" metric, which is based on covariance, but more numerically stable. Other transformations (e.g.: saturation or removing outliers) of covariance used in the reward function could also meet the intent of quantifying uncertainty, while avoiding the pitfall of numerical explosion. The second method is to ensure that the conditions in which filters diverge do not occur. This can be done by terminating an episode if any covariance matrix begins to explode, or setting the process noise Q, which governs the growth rate of covariance values, low enough such that none of the filters diverge during an episode.

Directly Useful Spatial Features in Observation Space

When designing the observation space for the SSA sensor allocation problem, a reasonable and obvious feature to include in the observation space is the dynamic state of all targets. Indeed, this is done in many of the papers in the literature review (see Sec. 2.2). Including the dynamic state gives the RL agent information about orbital dynamics. However, orbit dynamics are well known and there are features which provide more direct spatial information about the sensor-target topology than the ECI state of the targets.

For the scenario in which sensors have an unlimited FOR, the sensor-target topology is trivial; all sensors have access to all targets at all times. This scenario is covered in some of the papers discussed in Sec. 2.2. However, in the limited FOR case, discussed in this dissertation and other papers in Sec. 2.2, the topology is non-trivial and time dependent. In this case, the important information for the RL agent, with regards to the spatial relationship between sensors and targets, is whether or not a given sensor can access a given target. This information is recoverable from the ECI states of the sensor and target, but there is no practical reason for the RL agent to derive this information, which can be easily solved for.

An observation space that includes a direct representation of sensor-target topology is more computationally efficient than one that includes raw ECI state. Including sensor-target topology directly allows the RL agent to immediately learn the sensor allocation portion of the problem, not spending training iterations embedding orbit dynamics. Including any feature that directly reveals spatial relationships between sensors and targets is a more efficient methodology than including only the ECI dynamic states because it alleviates the need for the RL agent to estimate orbit dynamics. Some other examples of useful spatial features could include slant range, rate of change of azimuth angle relative to sensor, and angle from boresight (all of which are represented in Sec. 2.2). In this dissertation, the sensor-target topology is quantified via the "visibility matrix" Ω, first introduced in Ch. 3 and further refined in Ch. 5.

Design Initial Conditions to Ensure Non-Triviality

If the initial conditions of a SSA sensor allocation scenario are not purposefully designed, the simulations run the risk of not presenting the RL agent with sufficient non-trivial decisions to make. This concept is first introduced in Sec. 5.7. The initial condition design must ensure

that the sensor locations, RSO orbital parameters, horizon, and time step of the episodes are set such that there are many steps at which the sensor network has multiple, preferably many, targets to select from. The more non-trivial opportunities are presented to an RL agent, the more useful transitions the training algorithm can use when updating the weights of the model. A higher proportion of non-trivial opportunities is also useful in establishing benchmark performance. If a scenario provides too few non-trivial decision opportunities to the benchmark agent(s), the policies will appear to have similar performance, even if those policies have demonstrably different traits (e.g.: the random and greedy policies).

There is also a balance to strike in making the scenario too difficult for the RL agent to learn. Too many steps with many actions to select from can slow or inhibit the training process. A recommended practice is to decrease the difficulty of the initial conditions until the RL agent demonstrates an ability to learn, then incrementally increase the difficulty until the desired realism is achieved, or the training process becomes ineffective.

Some specific practices in setting initial conditions are as follows.

- For terrestrial sensors, ensure that the orbital planes of all RSOs pass over the sensors.
- Ensure that the horizon is large enough to guarantee that the sensors have a sufficient number (at least one) of access windows to the RSOs.
- Set the time step small enough such that short access windows are not skipped over when propagating the dynamics (assuming a discrete fixed time step propagator is used).
- Ensure that RSOs are spaced close enough such that multiple RSOs are frequently and simultaneously in the FOR of the same sensor. This can be done by restricting the bounds of the distributions of orbital parameters or increasing the number of RSOs.

Bibliography

[1] P. Anz-Meador, J. Opiela, and J.-C. Liou, "History of On-orbit Satellite Fragmentations, TP-20220019160, 16th Edition," tech. rep., Orbital Debris Program Office, NASA, 2022.

[2] T. J. Muelhaupt, M. E. Sorge, J. Morin, and R. S. Wilson, "Space Traffic Management in the New Space Era," *Journal of Space Safety Engineering*, vol. 6, pp. 80–87, June 2019.

[3] J. G. Miller, "A New Sensor Allocation Algorithm for the Space Surveillance Network," *Military Operations Research*, vol. 12, no. 1, 2007.

[4] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning with Applications in R*, pp. 403–404. Springer, 2 ed., 2021.

[5] S. Training and R. Command, "Space Doctrine Publication 3-100: Space Domain Awareness," tech. rep., United States Space Force, Nov. 2023.

[6] B. Weeden, P. Cefola, and J. Sankaran, "Global Space Situational Awareness Sensors," *Advanced Maui Optical and Space Surveillance Technologies Conference*, 2010.

[7] A. B. Poore, J. M. Aristoff, and J. T. Horwood, "Covariance and Uncertainty Realism in Space Surveillance and Tracking," Tech. Rep. AD1020892, Air Force Space Command Astrodynamics Innovation Committee, Defense Technical Information Center, Feb. 2016.

[8] S. J. Setty, P. J. Cefola, O. Montenbruck, and H. Fiedler, "Application of Semi-Analytical Satellite Theory Orbit Propagator to Orbit Determination for Space Object Catalog Maintenance," *Advances in Space Research*, vol. 57, pp. 2218–2233, May 2016.

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

[10] B. Weeden, "Space Situational Awareness Fact Sheet," May 2017.

[11] B. Lal, A. Balakrishnan, B. M. Caldwell, R. S. Buenconsejo, and S. A. Carioscia, "Global Trends in Space Situational Awareness (SSA) and Space Traffic Management (STM)," Tech. Rep. AD1123106, Institute for Defense Analyses, Mar 2018.

[12] A. O. Hero, D. Castañón, D. Cochran, and K. Kastella, eds., *Foundations and Applications of Sensor Management*, pp. 1–3. New York, NY: Springer, 2008th edition ed., Nov. 2007.

[13] T. G. Roberts, P. M. Siew, D. Jang, and R. Linares, "A Deep Reinforcement Learning Application to Space-Based Sensor Tasking for Space Situational Awareness," *Advanced Maui Optical and Space Surveillance Technologies Conference*, 2021.

[14] P. M. Siew and R. Linares, "Optimal Tasking of Ground-Based Sensors for Space Situational Awareness Using Deep Reinforcement Learning," *Sensors*, vol. 22, 2022.

[15] P. M. Siew, D. Jang, and R. Linares, "Sensor Tasking for Space Situational Awareness Using Deep Reinforcement Learning," *American Astronomical Society Meeting*, 2021.

[16] T. M. Mitchell, *Machine Learning*, pp. 1–2. McGraw-Hill, 1997.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning*, pp. 1–7. The MIT Press, 2 ed., 2020.

[18] J. Zhang, H. Yuan, and H. Dong, *Applications*, pp. 367–467. Springer, 2020.

[19] R. S. Sutton and A. G. Barto, *Reinforcement Learning*, pp. 421–453. The MIT Press, 2 ed., 2020.

[20] K. D. Julian and M. J. Kochenderfer, "Distributed Wildfire Surveillance with Autonomous Aircraft Using Deep Reinforcement Learning," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 8, pp. 1768–1778, 2019.

[21] J. Tromp and G. Farnebäck, "Combinatorics of Go," in *Computers and Games* (H. J. Van Den Herik, P. Ciancarini, and H. H. L. M. Donkers, eds.), vol. 4630, pp. 84–99, Springer Berlin Heidelberg, 2007.

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning*, pp. 321–337. The MIT Press, 2 ed., 2020.

[23] R. S. Sutton, "Learning to Predict by Methods of Temporal Differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[24] G. Tesauro, "Temporal Difference Learning and TD-Gammon," *Communications of the Association for Computing Machinery*, vol. 38, March 1995.

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

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-Level Control through Deep Reinforcement Learning," *Nature*, 2015.

[27] R. S. Sutton and A. G. Barto, *Reinforcement Learning*, pp. 195–196. The MIT Press, 2 ed., 2020.

[28] R. Bellman and S. Dreyfus, *Dynamic Programming*, vol. 33, pp. vii–xviii. Princeton University Press, 2010.

[29] D. R. Penn, K. Schroeder, and J. Black, "Invalid Action Masking for Deep Reinforcement Learning Applied to Space Situational Awareness," in *Astrodynamics Specialist Conference*, (Big Sky, MT), American Astronautical Society, 2023.

[30] E. Wan and R. Van Der Merwe, "The Unscented Kalman Filter for Nonlinear Estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pp. 153–158, IEEE, 2000.

[31] D. A. Vallado, *Fundamentals of Astrodynamics and Applications, 4th ed.*, pp. 777–799. Microcosm Press, 4th edition ed., 2013.

[32] R. Linares and R. Furfaro, "An Autonomous Sensor Tasking Approach for Large Scale Space Object Cataloging," *Advanced Maui Optical and Space Surveillance Technologies Conference*, 2017.

[33] A. O. Hero, D. Castañón, D. Cochran, and K. Kastella, eds., *Foundations and Applications of Sensor Management*, pp. 7–32. New York, NY: Springer, 2008th edition ed., Nov. 2007.

[34] R. Bellman and S. Dreyfus, *Dynamic Programming*, vol. 33, pp. 81–115. Princeton University Press, 2010.

[35] H. Zhang and T. Yu, *Taxonomy of Reinforcement Learning Algorithms*, pp. 125–133. Springer, 2020.

[36] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv*, August 2017.

[37] J. Zhang, H. Yuan, and H. Dong, *Introduction to Deep Learning*, pp. 3–46. Springer, 2020.

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

[39] C. Olah, "Understanding LSTM Networks," 2015. Accessed: 2024-03-01.

[40] Y. He, L. Xing, Y. Chen, W. Pedrycz, L. Wang, and G. Wu, "A Generic Markov Decision Process Model and Reinforcement Learning Method for Scheduling Agile Earth Observation Satellites," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, pp. 1463–1474, mar 2022.

[41] Y. Huang, Z. Mu, S. Wu, B. Cui, and Y. Duan, "Revising the Observation Satellite Scheduling Problem Based on Deep Reinforcement Learning," *Remote Sensing*, vol. 13, June 2021.

[42] H. WANG, Z. YANG, W. ZHOU, and D. LI, "Online Scheduling of Image Satellites Based on Neural Networks and Deep Reinforcement Learning," *Chinese Journal of Aeronautics*, vol. 32, pp. 1011–1019, Apr. 2019.

[43] L. Ren, X. Ning, and J. Li, "Hierarchical Reinforcement-Learning for Real-Time Scheduling of Agile Satellites," *IEEE Access*, vol. 8, pp. 220523–220532, 2020.

[44] A. Hadj-Salah, R. Verdier, C. Caron, M. Picard, and M. Capelle, "Schedule Earth Observation Satellites with Deep Reinforcement Learning," in *International Workshop on Planning and Scheduling for Space*, (Berkeley, California, USA), July 2019. arXiv: 1911.05696.

[45] H. S. Ahn, O. Jung, S. Choi, J. H. Son, D. Chung, and G. Kim, "An Optimal Satellite

Antenna Profile Using Reinforcement Learning," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 41, pp. 393–406, May 2011.

[46] X. Zhao, Z. Wang, and G. Zheng, "Two-Phase Neural Combinatorial Optimization with Reinforcement Learning for Agile Satellite Scheduling," *Journal of Aerospace Information Systems*, vol. 17, pp. 346–357, July 2020.

[47] L. Wei, Y. Chen, M. Chen, and Y. Chen, "Deep Reinforcement Learning and Parameter Transfer Based Approach for the Multi-Objective Agile Earth Observation Satellite Scheduling Problem," *Applied Soft Computing*, vol. 110, Oct. 2021.

[48] R. Linares and R. Furfaro, "Dynamic Sensor Tasking for Space Situational Awareness," *Advanced Maui Optical and Space Surveillance Technologies Conference*, 2016.

[49] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A Distributed Framework for Emerging AI Applications," *arXiv*, 2018.

[50] P. M. Siew, D. Jang, T. G. Roberts, and R. Linares, "Space-Based Sensor Tasking Using Deep Reinforcement Learning," *The Journal of the Astronautical Sciences*, vol. 69, November 2022.

[51] P. M. Siew, T. Smith, R. Ponmalai, and R. Linares, "Scalable multi-agent sensor tasking using deep reinforcement learning," in *Proceedings of the Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference*, p. 3, 2023.

[52] Q. R. Hardesty, *Autonomous Sensor Tasking for Space Situational Awareness Using Deep Reinforcement Learning*. PhD thesis, University of Arizona, 2018.

[53] A. Harvey, K. Laskey, and K.-C. Chang, "Machine Learning Applications for Sensor Tasking with Non-Linear Filtering," *Sensors*, vol. 22, no. 6, p. 2229, 2022.

[54] B. Oakes, D. Richards, J. Barr, and J. Ralph, "Double Deep Q Networks for Sensor Management in Space Situational Awareness," *arXiv*, May 2022.

[55] SAIC, "Space-track.org," 2022.

[56] S. Huang and S. Ontañón, "A Closer Look at Invalid Action Masking in Policy Gradient Algorithms," *arXiv*, May 2022.

[57] D. Ye, Z. Liu, M. Sun, B. Shi, P. Zhao, H. Wu, H. Yu, S. Yang, X. Wu, Q. Guo, Q. Chen, Y. Yin, H. Zhang, T. Shi, L. Wang, Q. Fu, W. Yang, and L. Huang, "Mastering Complex Control in MOBA Games with Deep Reinforcement Learning," in *Proceedings of the Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence*, vol. 34, pp. 6672–6679, Apr. 2020.

[58] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 178:7234–178:7284, 2020.

[59] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023.

[60] A. Zheng and A. Casari, *Fancy Tricks With Simple Numbers*, ch. 2. O'Reilly Media, Inc., 2018.

[61] V. Tech, "Advanced research computing," 2022.

[62] D. Hendrycks, N. Carlini, J. Schulman, and J. Steinhardt, "Unsolved Problems in ML Safety," *CoRR*, vol. abs/2109.13916, 2021.

[63] J. A. Lawton, "Numerical Method for Rapidly Determining Satellite-Satellite and Satellite-Ground Station In-View Periods," *Journal of Guidance, Control, and Dynamics*, vol. 10, pp. 32–36, Jan. 1987.

[64] N. Salkind, *Encyclopedia of Research Design*, p. 936. SAGE Publications, Inc., 2010.

[65] E. Parisotto, F. Song, J. Rae, R. Pascanu, C. Gulcehre, S. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury, M. Botvinick, N. Heess, and R. Hadsell, "Stabilizing Transformers for Reinforcement Learning," in *Proceedings of the 37th International Conference on Machine Learning*, pp. 7487–7498, PMLR, Nov. 2020. ISSN: 2640-3498.

[66] G. Gundersen, "Entropy of the Gaussian," 2020. (Accessed on: 1 April 2024).

Appendices

Appendix A

Information Gain for Multivariate Normal Distribution

This is a derivation of the information gain for the multivariate normal (Gaussian) distribution [66]. Entropy is defined as

$$H(x) = -\int p(x) \log p(x) \mathrm{d}x \tag{A.1}$$

where $p(x)$ is the probability of event x occurring. The units of entropy depend on the base of the log operator. If the base is 2, the units are bits (aka shannons). If the base is 10, the units are dits (decimal units of entropy). If the base is e, the units are nats (natural units of entropy). Any base can be used provided it is consistent throughout the calculation. Let x be a multidimensional random variable with a Gaussian distribution,

$$x \sim N_D(\mu, P) \tag{A.2}$$

where D is the dimension of x, μ is the mean, and P is the covariance matrix. The entropy

of x is

$$H(x) = -\int p(x) \log p(x) \mathrm{d}x \tag{A.3a}$$

$$= -\mathbb{E}[\log p(x)] \tag{A.3b}$$

$$= \frac{D}{2}(1 + \log(2\pi)) + \frac{1}{2} \log |P| \tag{A.3c}$$

where $|\cdot|$ is the determinant. Equation (A.3b) is the definitions of the expectation operator. Equation (A.3c) is derived with much complication and use of properties of the trace operator. The difference in entropy between two distributions, also called the information gain, x_0 and x_1 is

$$H(x_0) - H(x_1) = \big[\frac{D}{2}(1 + \log(2\pi)) + \frac{1}{2} \log |P_0|\big] - \big[\frac{D}{2}(1 + \log(2\pi)) + \frac{1}{2} \log |P_1|\big] \tag{A.4a}$$

$$= \frac{1}{2} \log |P_0| - \frac{1}{2} \log |P_1| \tag{A.4b}$$

where the terms with D have cancelled out (note that both distributions have the same dimensions). Using the quotient of a logarithm identity, the information gain is

$$H(x_0) - H(x_1) = \frac{1}{2} \log \left(\frac{|P_0|}{|P_1|} \right). \tag{A.5}$$

Appendix B

Action Masking Study Supplemental Data

B.1 Greedy Policy Single Episode Actions

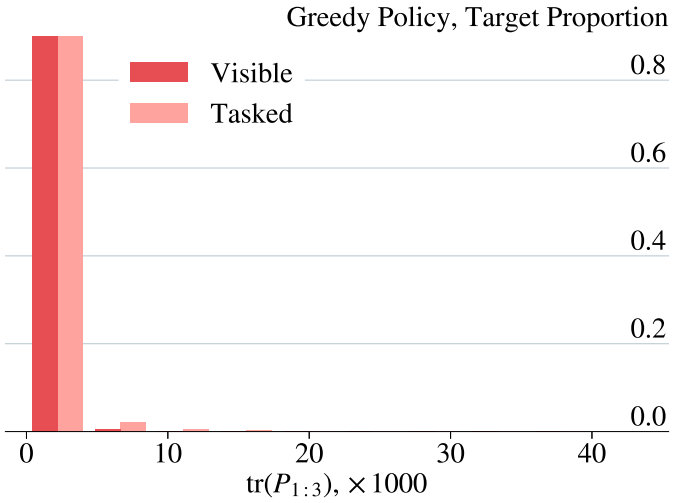


Figure B.1: Distribution of $\mathrm{tr}(P_{1:3})$ for visible and tasked targets for single episode of greedy policy. Distributions are normalized to their respective totals ($|\mathcal{N}_{\mathrm{vis}}| = 3651$, $|\mathcal{N}_{\mathrm{tasked}}| = 846$).

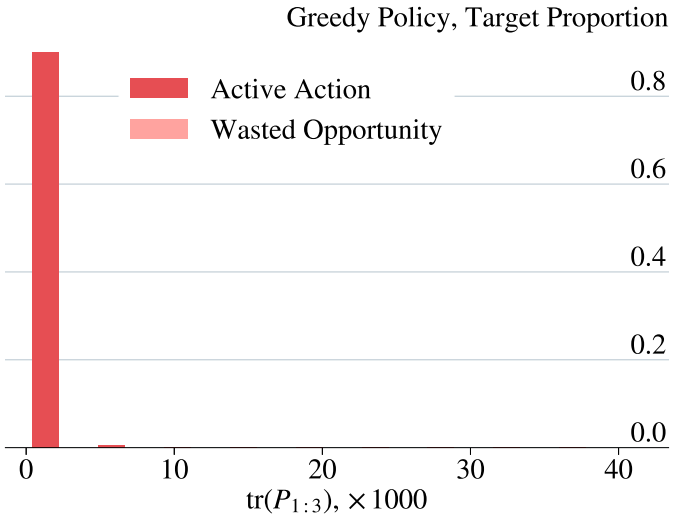


Figure B.2: Distribution of $\mathrm{tr}(P_{1:3})$ for active actions and wasted opportunities for single episode of greedy policy. Distributions are normalized to their respective totals ($|\mathcal{N}_{\mathrm{active}}| = 3526$, $|\mathcal{N}_{\mathrm{wasted}}| = 0$).

B.2 Monte Carlo Zoomed-In Box Plots

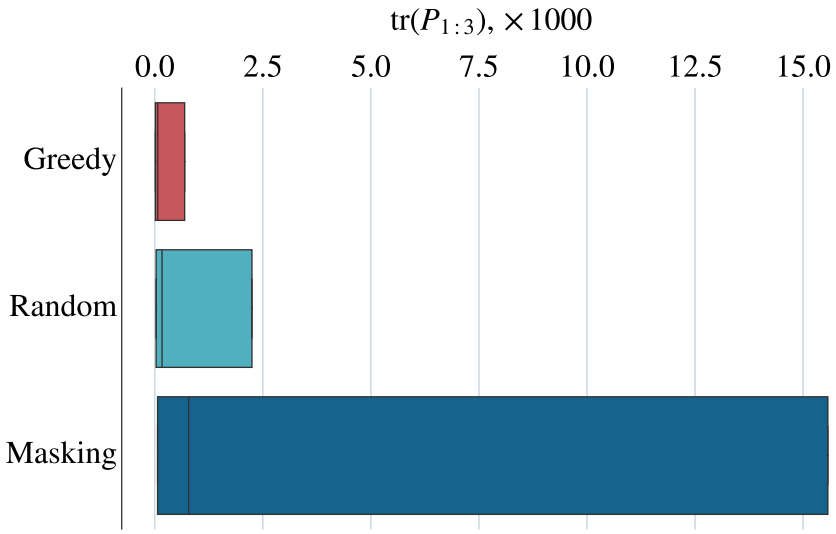
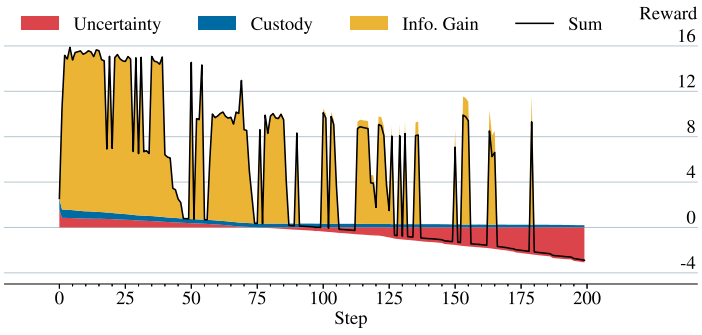


Figure B.3: Distribution of $\mathrm{tr}(P_{1:3}(T))$ for Monte Carlo experiment (lower is better; 100 episodes). Selected policies shown. Q1, median, and Q3 are shown. Whiskers omitted for clarity.

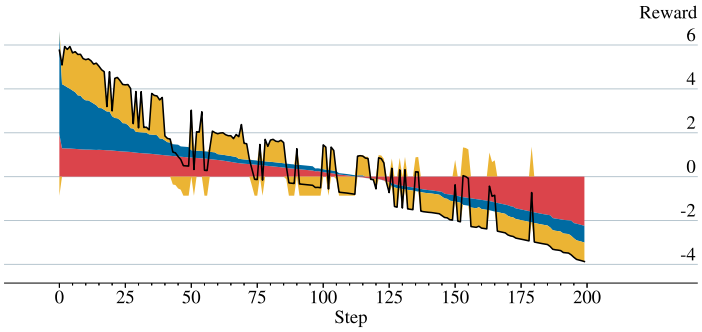
Appendix C

Environment Encoding Study Supplemental Data

C.1 Reward Function Decomposition, Greedy Policy



(a) Raw



(b) Normalized

Figure C.1: Time histories of raw and normalized reward function components (greedy policy).

C.2 Reward Function Coefficient Sensitivity

Table C.1: Nominal reward function coefficients (same as in Sec. 6.1). Unlisted orbital elements are 0.

(a) Environment parameters

Parameter	Value
M	1
N	100
T	200
SMA	$U(R_{\mathrm{E}}+300, R_{\mathrm{E}}+800)$
inclination	$U(-\pi/8, \pi/8)$
RAAN	$U(0, 2\pi)$
AL	$U(0, 2\pi)$

(b) Reward function coefficients

i	$\mathbf{m_i}$	$\mathbf{s_i}$
1	-0.59	1.11
2	0.31	0.15
3	4.45	5.13

Table C.2: Alternative reward function coefficients. Unlisted orbital elements are 0.

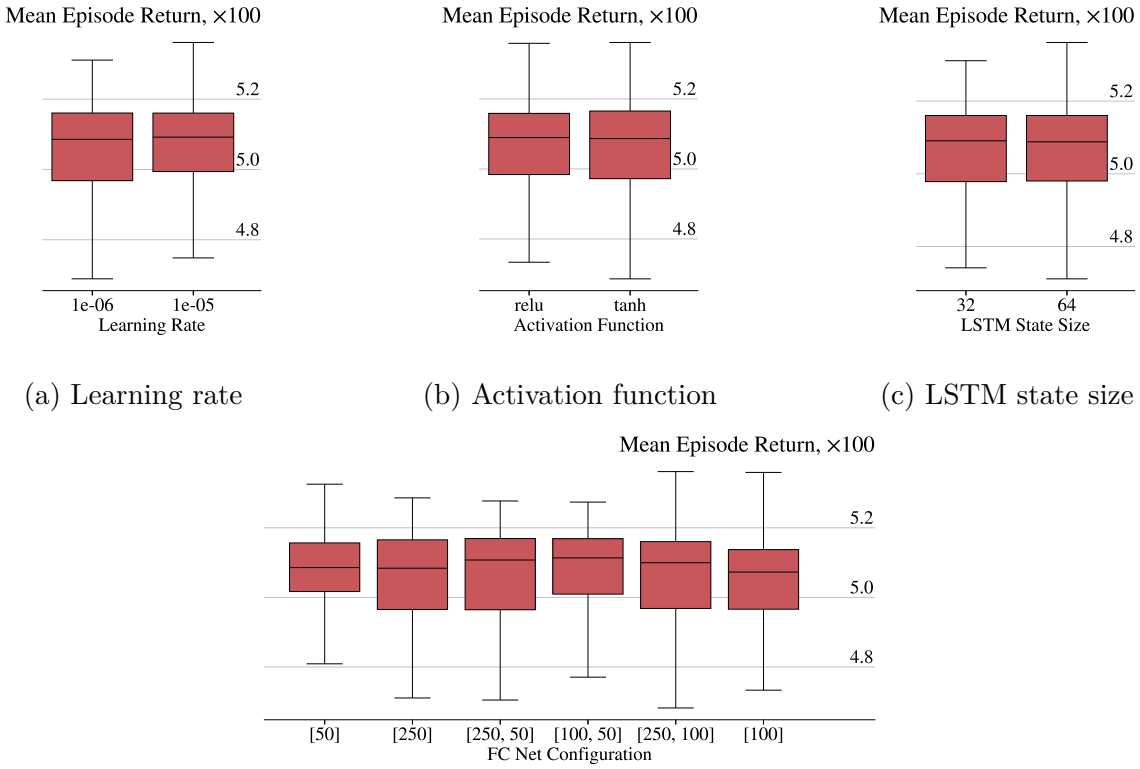
(a) Environment parameters

Parameter	Value
M	1
N	200
T	200
SMA	$U(R_{\mathrm{E}}+300, R_{\mathrm{E}}+800)$
inclination	$U(-\pi/8, \pi/8)$
RAAN	$U(0, 2\pi)$
AL	$U(0, 2\pi)$

(b) Reward function coefficients

i	$\mathbf{m_i}$	$\mathbf{s_i}$
1	-0.63	1.09
2	0.24	0.15
3	5.52	5.22

C.3 Hyperparameter Sensitivity for Short Horizon



(a) Learning rate

(b) Activation function

(c) LSTM state size

(d) FC network configuration

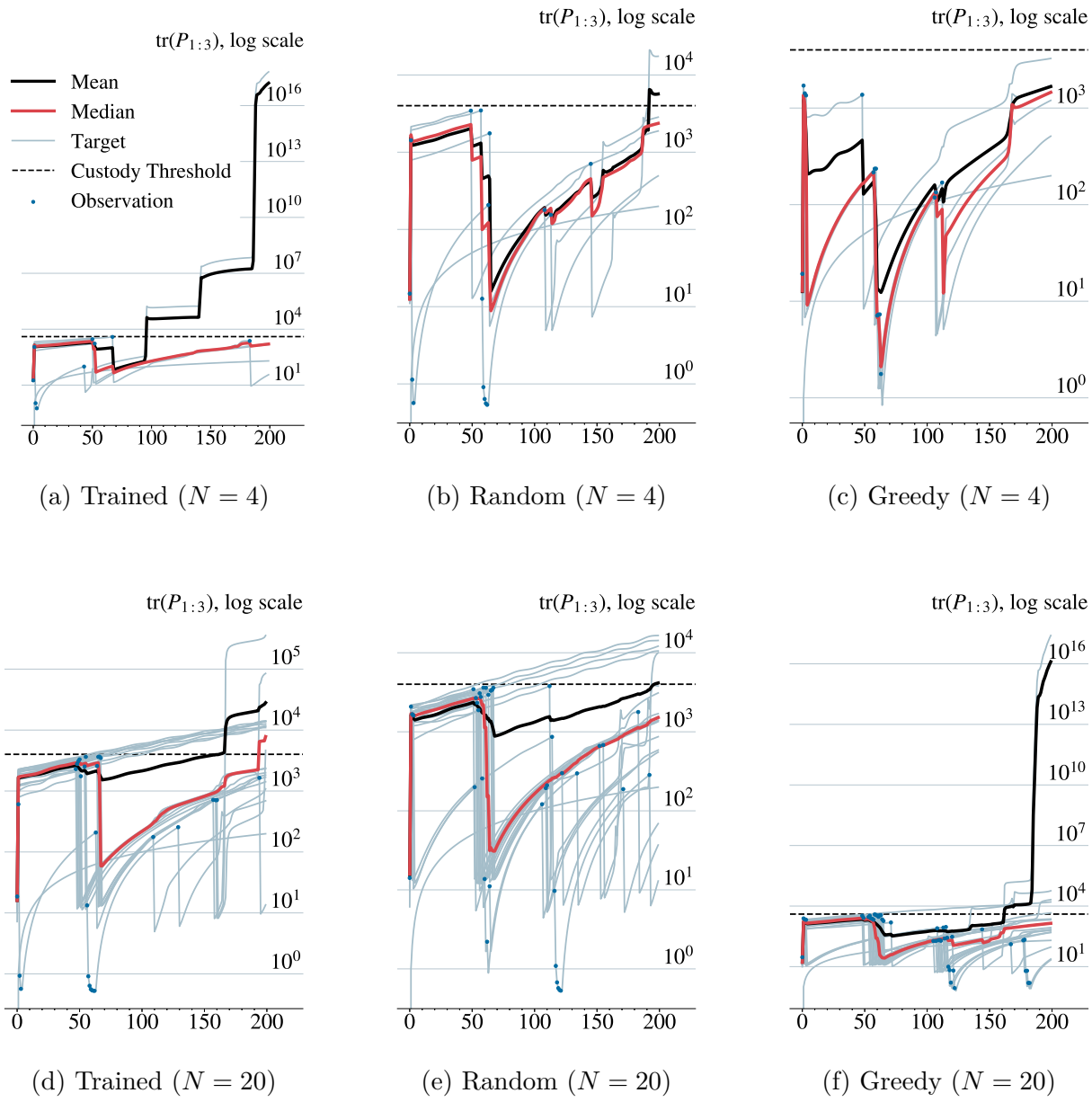
Figure C.2: Performance distributions for trained policies. Boxes shows quartiles ($T = 100$).

C.4 Best Policies for Short Horizon

Table C.3: Best-performing policies for short horizon.

Catalog Size	Learning Rate	Activation Function	FC Net Config.	LSTM Size	Mean Return	Cohort Mean Return
2	1×10^{-5}	tanh	[50]	64	514	509
3	1×10^{-5}	tanh	[250, 50]	64	517	511
4	1×10^{-6}	ReLU	[250, 100]	64	520	509
5	1×10^{-5}	ReLU	[100, 50]	64	523	511
6	1×10^{-5}	ReLU	[100, 50]	32	525	510
7	1×10^{-6}	ReLU	[100]	32	526	508
8	1×10^{-5}	ReLU	[250, 50]	32	527	511
9	1×10^{-6}	ReLU	[100, 50]	64	527	508
10	1×10^{-5}	tanh	[50]	64	533	511
15	1×10^{-5}	tanh	[250, 100]	64	536	511
20	1×10^{-5}	ReLU	[100]	64	536	507
25	1×10^{-6}	tanh	[100]	32	527	501
30	1×10^{-5}	ReLU	[100]	64	528	500
35	1×10^{-5}	tanh	[250, 50]	32	517	495
40	1×10^{-5}	tanh	[50]	32	524	490

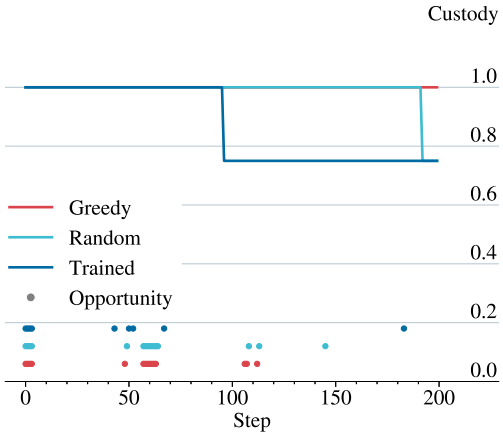
C.5 Supplemental Exemplar Episodes



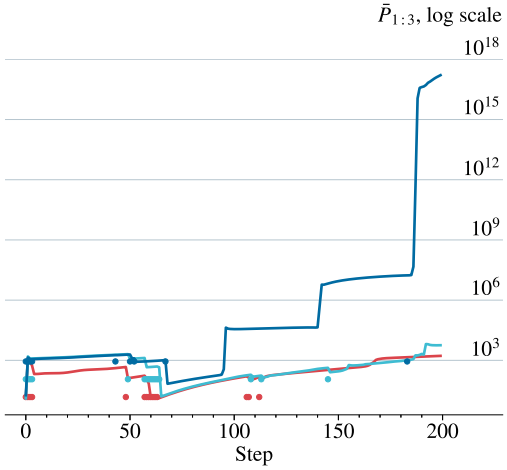
(a) Trained ($N = 4$) (b) Random ($N = 4$) (c) Greedy ($N = 4$)

(d) Trained ($N = 20$) (e) Random ($N = 20$) (f) Greedy ($N = 20$)

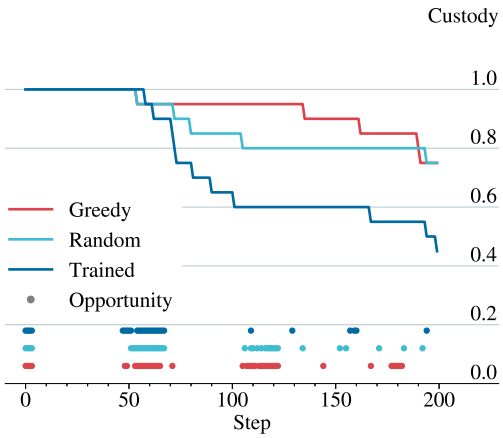
Figure C.3: Trace of positional covariance for single episode from Monte Carlo simulation (black: mean, red: median, gray: individual targets, dashed black: custody threshold, blue dot: observations).



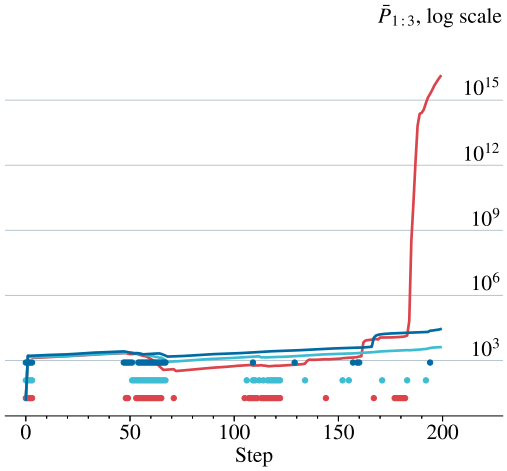
(a) Custody fraction ($N = 2$)



(b) Mean trace of positional covariance ($N = 2$)



(c) Custody fraction ($N = 20$)



(d) Mean trace of positional covariance ($N = 20$)

Figure C.4: Performance metrics for typical episode with all policies ($T = 200$).

C.6 Statistics of Non-Trivial Opportunities

Table C.4: Mean and STD of non-trivial opportunity fraction for Monte Carlo simulation ($N_{\text{ep}} = 50, T = 200$). Scaled to $100F_{\text{nt}}$.

Catalog Size	Trained		Random		Greedy	
	Mean	STD	Mean	STD	Mean	STD
2	2.36	0.82	2.23	0.70	2.42	1.77
3	2.76	1.10	3.19	1.35	3.51	1.79
4	3.80	1.65	4.11	1.84	4.53	1.95
5	4.50	1.43	4.54	1.99	4.98	2.10
6	4.70	1.39	5.75	2.17	5.72	1.80
7	5.26	1.71	6.94	2.23	6.11	2.27
8	6.53	1.66	6.68	1.59	7.58	2.43
9	7.54	1.75	7.04	2.09	8.41	1.81
10	7.14	2.08	8.05	2.37	8.04	2.64
15	8.67	1.79	10.57	2.61	12.23	2.43
20	10.27	1.78	12.78	2.58	14.08	2.02
25	12.05	2.40	14.13	2.13	16.32	1.81
30	12.06	1.72	14.77	2.64	18.05	1.68
35	14.02	2.35	16.04	2.30	18.94	1.81
40	14.76	2.37	17.13	2.46	19.37	1.81

C.7 Steps to 90% Custody

Table C.5: Statistics of $t_{0.9}$ for Monte Carlo simulation.

Catalog Size	Trained		Random		Greedy	
	Mean	STD	Mean	STD	Mean	STD
2	145	40	146	32	146	37
3	112	54	144	37	157	28
4	105	47	135	46	146	39
5	109	53	116	47	140	43
6	96	47	126	55	147	37
7	82	38	120	52	151	29
8	77	34	94	45	146	32
9	88	44	81	43	147	35
10	116	48	135	49	175	14
15	70	24	83	33	162	28
20	69	9	79	23	160	29
25	64	8	72	20	119	33
30	66	6	70	10	105	23
35	62	5	65	8	93	13
40	64	6	65	6	86	7