UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

RELATIONAL DEEP REINFORCEMENT LEARNING FOR AUTONOMOUS SENSOR CONTROL

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

Shane A. Flandermeyer
Norman, Oklahoma
2026

Downloaded from SHAREOK. Persistent Handle URL: https://hdl.handle.net/11244/342425

RELATIONAL DEEP REINFORCEMENT LEARNING FOR
AUTONOMOUS SENSOR CONTROL

A DISSERTATION APPROVED FOR THE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY THE COMMITTEE CONSISTING OF

Dr. Justin G. Metcalf, Chair

Dr. David S. Ebert

Dr. Andrew H. Fagg

Dr. Nathan A. Goodman

Dr. Dean F. Hougen

© Copyright by SHANE FLANDERMEYER 2026
All Rights Reserved.

To Mom, Dad, and Toast.

Acknowledgements

I would like to begin by thanking my advisor, Dr. Justin Metcalf. Throughout my academic career at OU he has been an incredible mentor and an even better friend. From fellowships to internships to interesting people, he has opened *so many* doors for me over the past seven years. I honestly did not plan to pursue a Ph.D. until I got the chance to work with him. I also appreciate Dr. David Ebert, Dr. Andrew Fagg, Dr. Nathan Goodman, and Dr. Dean Hougen for serving on my committee and providing valuable feedback at pivotal points in my doctoral degree. The principles I learned in their courses are the intellectual backbone of this work and my dissertation is stronger because of their efforts.

I am immensely grateful for the family members that have been with me throughout this journey. My mom and dad, Tracey Farrington and Doug Flandermeyer, have offered me more support than I could ever hope to repay. They've seen every high and low I've encountered along the way and have sacrificed so much to help me pursue my dreams. I also thank my dog, Toast, for being the definition of unconditional love. She didn't contribute anything meaningful to this work but she was a bigger part of it than she knows.

I am honored to be surrounded by more kind and intelligent friends than I can count. My time at the ARRC would've been infinitely more dull without Rylee Mattingly, Russell Kenney, Rachel Jarvis, Jon Knowles, Cal Schone, Geoff Dolinger, Alex Stringer, Tyler Reavis, Jonah Pehl, and so many others. Outside

the lab, I want to give special thanks to Nhi and Andrew Eudy, Duy Huynh, Callie Dosher, Jon Cason, Alica Alvarez, Derin Miller, Brian Smith, and Karissa Marion-Smith. Every single one of you means so much to me. Seriously.

Finally, I thank the National Science Foundation for supporting me through the graduate research fellowship program. The GRFP gave me the opportunity to explore fascinating research paths in my Masters and Doctoral studies that would be inaccessible otherwise. There is no doubt in my mind that I am a better scientist as a result.

Table of Contents

Abstract **xix**

1 Introduction **1**

1.1 Motivation 1

1.1.1 Cognitive Radar Spectrum Sharing 2

1.1.2 Autonomous Multi-object Search and Track 3

1.2 Prior Approaches and Related Work 4

1.2.1 Cognitive Radar Spectrum Sharing 4

1.2.2 Autonomous Multi-object Search and Track 6

1.3 Outline 8

2 Background **10**

2.1 Deep Learning 10

2.1.1 Attention Mechanisms 11

2.1.2 Recurrent Neural Networks 14

2.1.3 Relational Reasoning on Graph-Structured Data 16

2.1.4 Graph Neural Networks 20

2.1.5 Graph Network Architectures 21

2.2 Reinforcement Learning 24

2.2.1 Proximal Policy Optimization 29

2.2.2 Temporal-Difference Model Predictive Control 33
2.3 Single-Object Tracking 42
2.3.1 Tracking with Known Associations 46
2.3.2 Kalman Filtering 47
2.4 Multi-Object Tracking 55
2.4.1 Finite Set Statistics 56
2.4.2 Multi-Object Bayesian State Estimation 59
2.4.3 Track Oriented Multi-Bernoulli/Poisson Filter 60
2.5 Radar Signal Model 74

3 Cognitive Radar Spectrum Sharing 80
3.1 Proposed Algorithm 81
3.1.1 State Representation 81
3.1.2 State Encoder Network 84
3.1.3 Action Space 86
3.1.4 Reward Function 86
3.2 Experiment Setup 89
3.2.1 Hyperparameters and Baseline Comparisons 89
3.2.2 Evaluation Metrics 91
3.3 Results 93
3.3.1 2.4-2.5 GHz Environment 95
3.3.2 2.59-2.69 GHz Environment 105
3.4 Summary 110

4 Graph Learning for Autonomous Search and Track 112
4.1 Problem Formulation 112
4.1.1 Graph State Representation 113

4.1.2 Graph Encoder Network . . . 117
4.1.3 Reward Function . . . 119
4.2 Experiment Setup . . . 122
4.2.1 Baseline Comparisons . . . 122
4.2.2 Metrics . . . 124
4.2.3 Assumptions and Common Parameters . . . 126
4.3 Results . . . 128
4.3.1 Beam Optimization Task . . . 128
4.3.2 Mobile Surveillance Scenario . . . 138
4.4 Summary . . . 145

5 Conclusion **147**
5.1 Summary . . . 147
5.2 Novel Contributions . . . 148
5.3 Future Work . . . 150
5.3.1 Radar-Communications Spectrum Sharing . . . 150
5.3.2 Graph Learning for Sensor Control . . . 152

References **154**

A Preliminary Graph Search and Track Formulation **170**
A.1 Problem Formulation . . . 170
A.2 Proposed Algorithm . . . 171
A.2.1 Graph State Representation . . . 171
A.2.2 Entity Features . . . 173
A.2.3 Graph Encoder Architecture . . . 174
A.2.4 Reward Function . . . 176

A.3 Experimental Results 178
A.3.1 Search-Only Agent 179
A.3.2 Joint Search-and-Track Agent 180
A.4 Summary 183

B Acronyms and Abbreviations **185**

List of Tables

3.1 PPO agent parameters 90

3.2 Radar system parameters 90

4.1 Node features 113

4.2 Edge features 116

4.3 Beam Optimization Scenario Parameters 131

4.4 Search and track metrics for the beam optimization task. 134

4.5 Mobile Surveillance Scenario Parameters 141

4.6 Search and track metrics for the mobile surveillance task. 143

A.1 Node Features 174

List of Figures

2.1 Transformer encoder architecture. The model ingests a set of token feature vectors and optionally embeds positional information for sequential inputs. Next, the transformer applies Multi-head attention across tokens and an FFN network to each individual token. In the pre-LayerNorm configuration shown here, the *input* of each layer is normalized and the skip connection path remains unmodified. 13

2.2 Internal structure of a single LSTM cell. Orange and red boxes represent learned linear projections and nonlinear activation functions, respectively. Circles inscribed with "x" and "+" represent element-wise multiplication and addition operations. Converging lines indicate vector concatenation. 15

2.3 (Left) A simple example of a directed graph with global features. (Right) An attribute vector can be assigned to each individual node/edge/graph. In *homogeneous* graphs, all nodes and edges share a common feature space with all other nodes and edges, respectively. In *heterogeneous* graphs, the feature vectors of different node/edge classes need not share a vector space. 17

2.4 Graph representations of common machine learning building blocks. 18

2.5 TD-MPC2 world model components (Figure from [81]). At each time step, TD-MPC2 encodes the observed state into a sparse latent vector and predicts future rewards, values, and state transitions. TD-MPC2 does not use a decoder and performs all prediction and planning in the latent space. 34
2.6 Illustration of the Kalman update for a two-dimensional state/measurement space. The posterior mean is a convex combination of the measurement and prediction, where its location along the line is determined by the Kalman gain. 49
2.7 The unscented transform and its equations 50
2.8 Simple multi-object tracking scenario with a vehicle-mounted sensor. The tracker associated to the sensor must content with dynamic object appearance/disappearance, data association ambiguity, missed/false detections, and measurement noise. 56
2.9 Data association representation for an example scenario with three objects and four measurements. (Left) The example scenario. Colored ellipses represent the covariance of the object state estimates and black markers represent measurements. (Right) One possible association hypothesis for the scenario. The DA vector c_k is unique to the given hypothesis. 68
2.10 LFM waveform in the time domain for $B = 10\,\text{MHz}$ and $T_p = 10\,\mu\text{s}$. The blue curve represents the real component of the signal and the orange curve represents the imaginary component. 75
2.11 Frequency spectrum of a 10 MHz LFM waveform for various values of the time-bandwidth product. The spectral mask becomes asymptotically rectangular as the time-bandwidth product increases. 76

2.12 Pulse compression response for an LFM waveform. The matched filter produces a narrow peak at the received signal delay (the mainlobe) whose width is based on the range resolution of the radar. It also produces sidelobes that may mask the presence of other objects in the scene. 77

2.13 Range-Doppler response of an LFM after pulse-Doppler processing (pulse compression and a slow-time DFT) using $N_{CPI} = 32$ pulses. Applying pulse-Doppler processing to a coherent train of pulses produces a PSF with a peak at the range and Doppler of the object, but includes sidelobes in both dimensions. 78

3.1 Cognitive radar operational environment. The pulsed radar system is a secondary user in a congested spectrum environment, so it must adapt its emissions to reduce mutual interference while maximizing detection and tracking performance. 82

3.2 Example spectrogram in the 2.4-2.5 GHz band. Red bounding boxes indicate portions of the frequency spectrum occupied by other users. Yellow lines indicate the widest open band available for occupation by a radar system. 83

3.3 Recurrent-attention neural network architecture for the cognitive agent. The full-resolution observations following each pulse are projected into a learned embedding space, then the multi-head attention module identifies patterns at intra-pulse timescales. This information is further compressed via a pooling operation, then passed to an LSTM module which learns high-level patterns across pulses. 85

3.4 HO-CAE/FSS detections in scenarios encountered during training. 94

3.5 Episodic training reward in the 2.4-2.5 GHz ISM band (no waveform adaptation penalty). 96
3.6 Spectrum sharing metrics in the 2.4-2.5 GHz ISM band (no waveform adaptation penalty). 97
3.7 Waveform adaptation metrics in the 2.4-2.5 GHz ISM band (no adaptation penalty). 98
3.8 Continuous control agent behavior for varying values of the collision penalty weight α_c. (a) When the penalty is small, the agent prioritizes bandwidth utilization and collides with many other users. (b) Moderate collision penalties encourage balanced behavior similar to a SAA policy while (c) large penalties encourage the agent to prioritize collision avoidance. 99
3.9 Episodic training reward in the 2.4-2.5 GHz ISM band (varying adaptation penalty). 101
3.10 Spectrum sharing metrics in the 2.4-2.5 GHz ISM band (non-zero adaptation penalty). 102
3.11 Waveform adaptation metrics in the 2.4-2.5 GHz ISM band (non-zero adaptation penalty). 103
3.12 Range-Doppler responses across various adaptation penalty values. Each response is representative of a typical 256-pulse CPI in the 2.4-2.5 GHz band. To balance the inherent trade-off between waveform adaptation and collision avoidance, the reward function parameters for each agent are linearly related with $\alpha_c = 30 \cdot (1 - \beta)$ and $\beta = \beta_{bw} = \beta_{f_c}$. 104
3.13 Episodic training reward in the 2.59-2.69 GHz band (no waveform adaptation penalty). 105

3.15 Waveform adaptation metrics in the 2.59-2.69 GHz band (no waveform adaptation penalty). 106
3.14 Spectrum sharing metrics in the 2.59-2.69 GHz band (no waveform adaptation penalty). 107
3.16 Episodic training reward in the 2.59-2.69 GHz band (varying adaptation penalty). 108
3.17 Spectrum sharing metrics in the 2.59-2.69 GHz band (varying waveform adaptation penalty). 109
3.18 Waveform adaptation metrics in the 2.59-2.69 GHz band (varying waveform adaptation penalty). 110

4.1 Representative illustration of the graph state representation. The scenario graph is composed of agent, search, and track nodes, where each node represents the state of an entity at a particular time. Entity nodes are connected via state transition, update, or predict edges. 114
4.2 Graph processing network architecture. The network first projects each node, edge, and global feature vector into a shared latent space. Next, the model forms a graph and performs L layers of GNN processing on the entire graph. The decoding stage extracts the latent representation of the agent node from the current timestep to represent the complete graph state. 118
4.3 Reward heatmaps over several values of the task priority weights. Each image shows the reward landscape as a function of the undetected intensity weight (x-axis) and total track quality (y-axis). Red indicates high-reward regions while blue indicates low-reward regions. 123

4.4 Example realization of the beam optimization scenario. The stationary sensing platform (yellow circle) controls an electronically-steered beam (red sector). In this instance, the sensor maintains 4 active tracks (green circles) and has yet to detect one ground truth object. 128

4.5 Learning curves for the beam optimization task over 20 random seeds. The black line gives the average reward of the MCTS planning agent from [42] over 100 Monte Carlo episodes. The RL agent obtains similar task performance as the planner after training is complete. 132

4.6 Search and Track performance metric for the beam optimization task. Three configurations of the RL agent (search+track, search-only, track-only) are compared to the exhaustive planning agent from [42] in terms of (a) GOSPA, (b) mean localization error, (c) Miss error, (d) False error, (e) Total undetected intensity weight, and (f) Total track quality. 133

4.7 Scaling performance of the encoder network as a function of the input graph size (assuming 3 edges per node). Runtime complexity scales linearly with the size of the graph, and the encoder maintains sub-millisecond inference time for graphs with fewer than 10^4 nodes. 135

4.8 Exhaustive planner runtime analysis. The computational requirements of the exhaustive planner from [42] scale exponentially with the planning horizon (colored lines). The graph RL agent maintains a constant runtime of 1.2 ms per decision during inference (dashed black line). 136

4.9 One realization of the mobile surveillance scenario. The sensing platform (yellow circles) maintains tracks (green circles) on two ground truth objects (red stars) and has yet to detect a third object in the bottom half of the surveillance region. For visual clarity, older states in the platform trajectory have smaller circle markers than more recent states. Gray lines represent prediction edges between agent nodes and search nodes. 138

4.10 Learning curves for the mobile surveillance task over 20 random seeds. The black line gives the average reward of the MCTS planning agent from [42] over 100 Monte Carlo episodes. The RL agent ultimately achieves a higher reward than MCTS after training is complete. 142

4.11 Search and Track performance metric for the mobile surveillance task. Three configurations of the RL agent (search+track, search-only, track-only) are compared to the MCTS planning agent from [42] in terms of (a) GOSPA, (b) mean localization error, (c) Miss error, (d) False error, (e) Total undetected intensity weight, and (f) Total track quality. 144

4.12 Planning runtime complexity vs planning horizon. 145

5.1 Proposed multi-sensor extension to the graphical search and track approach. Two autonomous platforms (agents 1 and 2) maintain a shared graph state. The multi-agent formulation extends the single-agent graph with separate trajectory nodes for each sensor and inter-agent communication edges (blue). Search nodes are omitted for clarity. 153

A.1 Graph representation of the multi-object search-and-track environment. The scenario graph is composed of agent, search, and track nodes, which can be connected through various edge types. Search nodes act as spatial aggregators which share information between their k nearest neighbors. 172

A.2 General encoder architecture for the autonomous agent. Each node type is processed by a separate 2-layer MLP before forming the graph, and the graph is processed with L graph neural network (GNN) layers. The final feature vector is the vector concatenation of the current agent node and pooled node features. 175

A.3 Episodic training reward for RL agents with (a) no track maintenance requirements and (b) a maximum tolerable localization error of 15 m. In both cases, the RL agent learns to outperform its heuristic counterparts. 181

A.4 Search and track metrics for the RL agents over a representative episode. The search-and-track agent maintains significantly lower localization error than the search-only agent with minimal impact to the search performance. 182

Abstract

Rapid improvements in processing hardware and sensing technologies have enabled the development of autonomous systems across countless domains. To fully realize the potential of these systems, sensor control algorithms must efficiently leverage limited sensing resources to meet task objectives while satisfying latency constraints for real-time operation. Traditional techniques based on optimal control theory and classical signal processing often struggle to meet these demands due to their high computational requirements and poor scalability. Existing learning-based techniques are computationally lightweight but frequently fail to generalize in unseen scenarios. Instead, this dissertation proposes that deep reinforcement learning (RL), relational reasoning, and continuous control are powerful tools that can be combined to solve a large class of sensor control problems. This hypothesis is explored in two increasingly relevant application spaces: spectrum allocation between radar and communications devices and sensor management for multi-object search and track tasks. In both cases, the relational reasoning techniques outperform state-of-the-art methods from the literature in terms of task performance, runtime complexity, or both.

The first case study focuses on the radio frequency (RF) spectrum sharing problem from a radar perspective. The growing demand for RF spectrum has placed considerable strain on radar systems. Future radar systems must be designed with coexistence in mind to avoid harmful mutual interference that com-

promises the quality of service for other users in the channel. This work presents a deep RL approach to spectrum sharing that enables a pulse-agile cognitive radar to operate in congested spectral environments. The deep RL approach is shown to outperform several techniques from the literature while generalizing across diverse scenarios in a software-defined radio (SDR) testbed.

The second half of this dissertation focuses on sensor control for the multi-object search and track problem. In these tasks, an autonomous platform must search a region of interest for unknown objects while maintaining accurate state estimates on detected objects. Existing solutions to this problem rely on computationally intensive online planning routines that scale poorly to complex, high-dimensional control spaces. This work develops a graph-theoretic solution that leverages recent advances in relational reasoning and model-based RL. Experimental results demonstrate that the graph RL approach outperforms state-of-the-art planning techniques in terms of tracking performance while reducing execution times by several orders of magnitude. This makes the proposed approach highly suitable for real-time operation on resource-constrained platforms.

Chapter 1

Introduction

1.1 Motivation

In autonomous sensing scenarios, one or more activate sensing platforms use feedback from their environment to update their control parameters to meet task objectives. This general formulation encompasses domains ranging from multi-function radar task scheduling [1], [2] to simultaneous localization and mapping (SLAM) on mobile robotic platforms [3]. This dissertation focuses on two application spaces that have become highly relevant in recent years: dynamic radar-communications spectrum allocation and sensor control for multi-object search and track. Existing techniques in these areas use either myopic single-step heuristics, multi-step planning, or machine learning to select optimal sensing actions. These techniques must navigate a rich trade space between computational complexity, generalization to novel scenarios, and overall performance on the task at hand. The primary goal of this work is to demonstrate that deep reinforcement learning (RL), relational reasoning, and continuous control provide a powerful framework for solving autonomous control problems. As will be shown, autonomous agents trained using these principles are well-suited for real-time operation in dynamic environments and can generalize across a variety of

operating conditions with minimal hand-tuning. The remainder of this section describes the individual application spaces in greater detail and outlines prior approaches in the existing literature.

1.1.1 Cognitive Radar Spectrum Sharing

Access to the radio frequency (RF) spectrum is essential for countless systems and services. Each application space desires large bandwidths to maximize its quality of service. For instance, wireless communications networks require large bandwidths to support high data rates for a growing user base, while radar systems use wideband waveforms to improve their range resolution and obtain a finer radar image. However, the RF spectrum is a scarce resource that is becoming increasingly congested as wireless networks evolve to support an ever-growing number of users and devices [4]–[7]. Consequently, pressure has mounted from the telecommunications industry to reallocate existing spectrum previously assigned to radar for communications in the sub-6 GHz bands [8], [9].

In tandem with this regulatory effort, several initiatives have proposed dynamic spectrum access (DSA) techniques to efficiently share the spectrum among systems with disparate requirements and operating behaviors [10]. Cognitive radar and radio specifically have been identified as key enabling technologies for DSA and spectrum sharing [11]. A cognitive radar must analyze the spectrum in real time and modify its emissions to avoid mutual interference with other users. This work develops a control framework for spectrum sharing using a cognitive radar that adapts its emissions on a pulse-by-pulse basis. The proposed approach trains an autonomous agent to operate a pulse-agile radar in congested spectrum environments without interfering with other users.

1.1.2 Autonomous Multi-object Search and Track

This dissertation also investigates the problem of jointly searching for and tracking multiple objects using an active sensor. A search-and-track system must maintain accurate state estimates for detected objects under track while continuously surveilling the region of interest for objects that have not yet been detected. This is a critical task for applications such as autonomous driving [12], search-and-rescue missions [13], and unmanned aerial system (UAS) surveillance [14]. In many cases the sensing platform is heavily limited in its sensing and computational capabilities. Therefore, decision making algorithms deployed on these platforms must be computationally lightweight and while operating under constraints that may vary across systems or scenarios. Ideally, these algorithms should seamlessly generalize transfer to different sensing modalities (e.g., radar or lidar) and platform types (e.g., ground vehicle or aerial vehicle).

The joint search and track problem is complicated by several factors. First, the search and track tasks often conflict with one another and compete for the limited resources of the active sensor(s). Autonomous agents must therefore dynamically prioritize each task based on system requirements and the current environment state. Practical sensors are imperfect, so agents must operate within a limited field-of-view (FOV) while contending with noise, false alarms, and missed detections. When the number of tracked objects is unknown and time-varying, the agent also needs to model object appearance and disappearance while resolving data association ambiguities between measurements and tracked objects. This modeling is typically performed in a multi-object tracking filter.

Recently, the multi-object tracking community has devoted significant attention to state estimation techniques based on random finite set (RFS) statistics. RFS filters model both object states and measurements as time-varying finite

sets, providing a provably Bayes optimal representation of the multi-object posterior state distribution [15], [16]. Since the optimal multi-object Bayes filter quickly becomes computationally infeasible in practice, several tractable approximations have been developed in the literature. Some examples include the probability hypothesis density (PHD) [17], Poisson multi-Bernoulli mixture (PMBM) [18], and track-oriented marginal Bernoulli/Poisson (TOMB/P) filters [19], [20]. The various RFS filters differ primarily in their representation of the multi-object state density and in their computational requirements. For instance, the PMBM filter maintains a hypothesis tree similar to classical multiple hypothesis tracking (MHT) [21] while the TOMB/P filter marginalizes over data association hypotheses to produce a single state estimate for each hypothesized object [22]. As will be shown, the TOMB/P and PMBM filters also provide useful state representations for the search task that can be exploited by autonomous control algorithms.

1.2 Prior Approaches and Related Work

1.2.1 Cognitive Radar Spectrum Sharing

Several radar-centric cognitive approaches to DSA have been developed in prior works. Early approaches tended to be purely reactive to the current environment state without planning over future time steps. For example, [23] and [24] advocate for a sense-and-avoid (SAA) approach for pulse-agile radar operation. Under the SAA paradigm, the radar first identifies unoccupied frequency bands using an energy detection techniques such as the fast spectrum sensing (FSS) algorithm from [25]. The radar uses the spectrum occupancy information to select waveform parameters that optimize metrics such as the waveform bandwidth and signal-to-interference-plus-noise ratio (SINR). The authors of [26] take an alternative

approach, using the FSS algorithm to identify occupied portions of the spectrum, then adaptively notching a base radar waveform in the occupied bands.

Reinforcement learning has been explored for several applications in the cognitive radio literature, including spectrum access, wireless power control, and data rate adaptation [27], [28]. The work in [29] is one of the first approaches that applies deep RL to the DSA problem, using a deep Q-network (DQN) to learn to access a dynamic channel with unknown occupancy statistics. The approach in [30] uses a deep recurrent Q-network (DRQN) to learn a robust channel access strategy for a secondary user (SU) operating in the presence of multiple heterogeneous primary users (PU), achieving lower collision rate and higher access rate than myopic policies from the literature. The authors of [31] and [32] examine the spectrum access problem for multiple cooperative users, utilizing multi-agent reinforcement learning (MARL) techniques and DRQNs to learn a cooperative strategy that maximizes network throughput.

Deep RL methods for wireless power allocation have been formulated in both discrete and continuous control domains. For example, the authors of [33] propose a novel learning framework to allocate power between a set of base stations, achieving greater energy efficiency compared to methods based on deep deterministic policy gradient (DDPG) [34] and other traditional power allocation methods. Similarly, [35] uses proximal policy optimization (PPO) to efficiently allocate power between SUs while maintaining quality-of-service requirements for both the PU and the SUs. The approach in Chapter 3 solves the converse of the power allocation problem: rather than allocating a *continuous* power value for a *discrete* number of channels, the proposed approach uses RL to allocate a *binary* power value (transmit or not transmit) to a *continuous* range of frequencies.

RL-based DSA approaches have also recently been explored from a radar per-

spective. For example, [36] and [37] use the policy iteration algorithm to avoid interference while tracking a moving target. The authors of [38] and [39] present a deep RL extension to this work, using common variants of the DQN (double DQN and DRQN) to perform the same task in more complex scenarios, showing that deep methods can successfully generalize in time-varying, dynamic environments. In [40] and [41], the authors formulate waveform selection as a linear contextualized bandit problem to facilitate sample-efficient online learning. They also introduce a reward penalty for rapid waveform changes that is generalized in this work.

The cognitive radar DSA approaches described above have several limitations that preclude their use in practical systems. First, they formulate the spectrum sharing problem as a discrete control task with a small number of actions. As noted in [39], the number of feasible control inputs scales *exponentially* with the frequency resolution of the action space. This limits the flexibility of the agent and significantly degrades its decision-making capabilities in wideband channels. Though several of the techniques discussed above use recurrent architectures to model temporal channel dynamics, they typically operate on individual spectrum snapshots and do not sufficiently capture long-term behavior over entire radar processing intervals. Therefore, a major goal of this work is to develop scalable RL techniques for cognitive radar with high-level reasoning capabilities over long time horizons.

1.2.2 Autonomous Multi-object Search and Track

Sensor control techniques for the search and track problem have been a central focus in recent literature. The authors of [42] formulate the search and track task as a optimal control problem that is solved with an online planning al-

gorithm. For small action spaces and short optimization horizons, the planner exhaustively simulates all possible actions and selects the action that maximizes the objective. This is infeasible for larger problems, so the authors propose a planner that identifies performant actions via Monte Carlo tree search (MCTS) [43]. This technique can be easily adapted to various sensor types and scenarios but requires computationally intensive simulation during planning and heavily discretizes the control space to remain tractable. The work in [14], [44] considers the multi-sensor case using a similar planning approach. Their technique formulates the problem as a partially observable Markov decision process (POMDP) and performs an exhaustive search to identify the value-maximizing action sequence. Consequently, this approach has the same computational limitations as [42]. The authors of [45] use a probability hypothesis density (PHD) filter to plan the trajectories of a team of autonomous platforms. Though their approach is more computationally efficient than the methods above, they only consider the search task and do not jointly optimize for tracking performance.

Several previous works have also examined autonomous operation in simultaneous localization and mapping (SLAM) applications [46]. Though this work does not directly focus on the SLAM problem, both SLAM and multi-object tracking possess a sparse graphical structure that can be exploited for efficient optimization [19], [47]. The approach in [48] formulates the SLAM problem as a POMDP and uses a gradient-based planner to select actions from a continuous set of control inputs. Though this technique does not require discretization, the iterative nature of the planning procedure incurs significant computational overhead and precludes its use in real-time applications. Similar to our work, the authors of [49] represent the SLAM problem as a graph and train deep RL agents to solve an active SLAM task. The RL SLAM approach is highly scalable

and computationally efficient, but their graph formulation and reward structure are specific to SLAM applications and do not generalize to the search and track tasks of interest in this work.

1.3 Outline

This dissertation develops several RL techniques for autonomous sensor control. These approaches are designed to meet the stringent requirements for real-time operation and use relational learning techniques to generalize across diverse scenarios with minimal modification. The document is organized as follows. After the introduction in this section, Chapter 2 provides the mathematical background that underlies the techniques derived in later chapters. The background begins with a general discussion of reinforcement learning, followed by discussions of the specific algorithms (PPO and TD-MPC2) used in this work. The discussion proceeds to discuss the relevant deep learning modules (attention, RNNs, and GNNs). Next, the chapter defines the single- and multi-object state estimation techniques used for the tracking component of this work, including the family of Kalman filters and random finite set trackers. The chapter concludes by defining a signal model for a pulse-agile radar system that extracts range and velocity information using pulse-Doppler processing.

Chapter 3 presents a deep RL approach for pulse-agile radar operation in congested spectrum environments. The chapter describes the proposed POMDP formulation of the problem, including the state representation, action space, and reward function, along with a novel recurrent-attention architecture to process the spectrum observations with low latency. The chapter concludes with several experiments using real spectrum data from the 2.4-2.5 GHz and 2.59-2.69 GHz industrial, scientific, and medical (ISM) bands. These experiments show that the

RL agent can effectively operate in congested spectrum environments and can adapt the proposed reward function to obtain a wide range of agent behaviors.

Chapter 4 formulates a graph-theoretic approach for autonomous multi-object search and track using model-based RL. The chapter begins by describing the graphical structure inherent to the class of search and track problems, then defines a graph encoder network architecture and reward function that can be used by RL multi-object tracking agents. The graph RL technique is analyzed in terms of its runtime computation requirements, search/track task performance, and learning efficiency. Multiple simulation studies show that the proposed agent outperforms state-of-the-art planning techniques from the literature.

Chapter 5 concludes the work and provides several promising avenues for future work.

Chapter 2

Background

2.1 Deep Learning

Machine learning methods have improved significantly in recent years, attaining state-of-the-art performance in domains such as language modeling, computer vision, and autonomous control. Due to the widespread availability of low-cost computational resources and internet-scale datasets, much of the deep learning community has gravitated towards large foundational models that learn entirely from data. However, this purely data-driven approach is not feasible in sensor control applications with limited training data and tight computational constraints. This dissertation therefore develops learning techniques for sensor control that maximally exploit the problem structure and use relational reasoning to reduce computation and improve learning efficiency. This section discusses the deep learning building blocks used within the autonomous agents discussed in Chapters 4, including attention mechanisms, recurrent neural networks, and graph networks.

2.1.1 Attention Mechanisms

Attention mechanisms are powerful building blocks for relational reasoning that have become ubiquitous in modern deep learning following the success of the transformer architecture [50]. Though there are countless definitions of attention, this work defines attention as a weighted average over a set of input elements with dynamic weights that depend on the input data. This dynamic weighting allows the attention mechanism to selectively focus on the portions of the input that are most relevant to the task at hand.

The most common form of attention is the scaled dot-product attention operator defined in [50]. Dot-product attention operates on *tokens*, which are feature vectors that represent individual entities in the problem at hand. For example, language models commonly treat each individual word in a body of text as a token. Given a set of N input tokens, dot-product attention maps each token to a learned query, key, and value vector and packs them into matrices Q, K, and V, respectively. Scaled dot-product attention is then computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{2.1}$$

The dot product measures the pairwise similarity between each query and key vector. The softmax operation produces normalized attention scores which sum to unity for each query vector and whose values depend on the dot product QK^T. The attention output is a weighted average of the value vectors in the V matrix.

Multi-head attention (MHA) extends the standard attention formulation by performing multiple attention computations in parallel [50]. The final output is

a linear projection of the concatenated outputs of each head:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O, \tag{2.2}$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$. To maintain similar parameter counts to single-head attention, each head reduces the dimensionality of the query, key, and value vectors by a factor of the number of heads h. Let d_q and d_v be the dimensionality of the query/key and value vectors, respectively. The learned parameter matrices have the shape $W_i^Q \in \mathbb{R}^{d_q \times d_q/h}$, $W_i^K \in \mathbb{R}^{d_v \times d_v/h}$, $W_i^V \in \mathbb{R}^{d_v \times d_v/h}$, and $W^O \in \mathbb{R}^{hd_v \times d}$. Standard implementations of MHA set $d_q = d_v = d$.

The dot-product attention operator in Equation (2.1) processes the input tokens as a *set* and its output does not depend on the ordering of the input tokens. This property is undesirable in sequence modeling tasks (e.g., language modeling) where the token ordering provides important contextual information. Therefore, the transformer from [50] adds a sinusoidal position encoding to each token to embed position information as:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos/s^{(2i/d)}\right), \text{ and} \\ PE_{(pos,2i+1)} &= \cos\left(pos/s^{(2i/d)}\right), \end{aligned} \tag{2.3}$$

where pos is the index of the token in the sequence, i is the index of the vector dimension, and s is a scaling factor.

The output of the dot-product attention operator is a convex combination of the input elements. To improve expressivity, the transformer introduces a three-layer feed-forward network (FFN) after each attention block that is applied independently to each output token. The original transformer architecture also

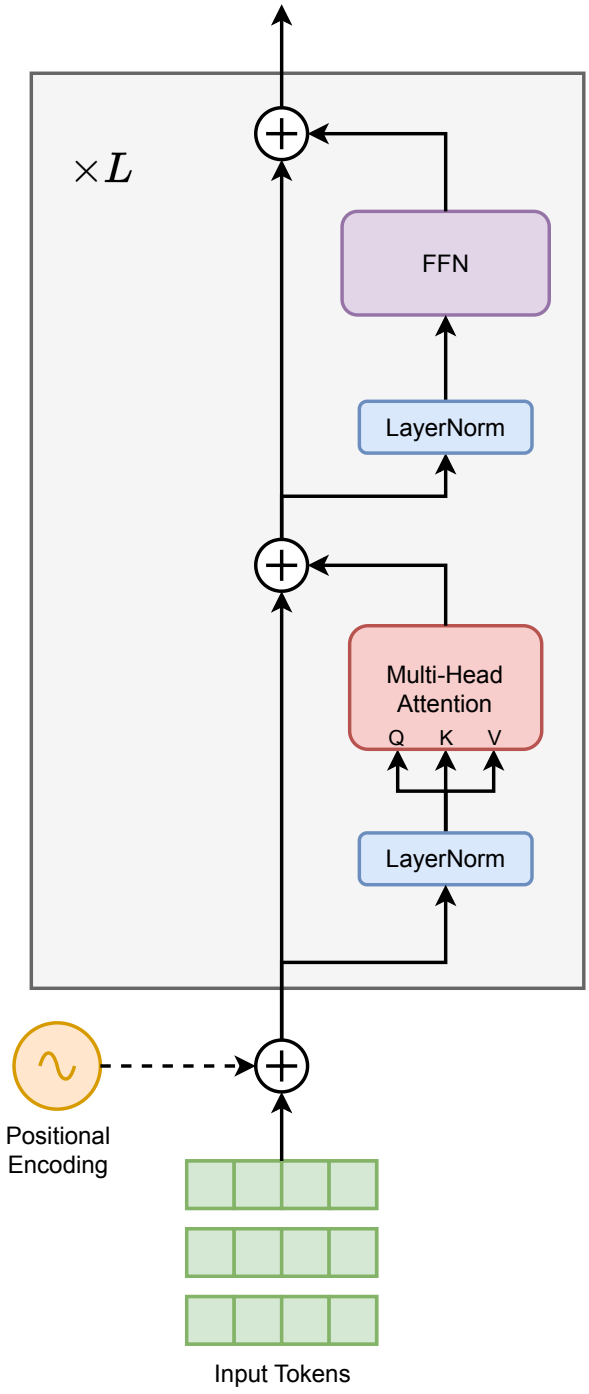


Figure 2.1: Transformer encoder architecture. The model ingests a set of token feature vectors and optionally embeds positional information for sequential inputs. Next, the transformer applies Multi-head attention across tokens and an FFN network to each individual token. In the pre-LayerNorm configuration shown here, the *input* of each layer is normalized and the skip connection path remains unmodified.

applies skip connections and Layer Normalization (LN) [51] after each MHA/FFN block, but recent work has shown that applying the LN *before* each processing step improves training stability and learning efficiency [52]. Figure 2.1 illustrates the encoder block for a pre-LN transformer. By stacking multiple encoder blocks, the transformer can learn complex, higher-order relationships between entities. Although the original transformer architecture also includes a decoder component for sequence-to-sequence tasks, the network architectures in this dissertation use only the encoder portion.

Figure 2.1 shows the internal structure of a pre-LN transformer encoder block. For simplicity, the figure illustrates self-attention, where the query, key, and value vectors are computed from a single set of input tokens. The dashed line in the positional encoding step indicates that it is only required if the input values represent a sequence. The encoder block transforms a set (or sequence) of input queries to an output set of latent vectors of the same size. Each output vector corresponds to an input query vector after it has been processed by the attention and FFN components.

2.1.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a class of models designed to process sequential data. RNNs maintain a hidden latent state that is updated at each time step based on the current input and previous hidden state. The learned parameters of this state update are shared across time steps, so RNNs are invariant to temporal translations in the input data. In many sequence modeling tasks, RNNs have been supplanted by transformer-based architectures due to their scalability and performance on large datasets [53], [54]. However, transformer models require relatively large amounts of training data and scale quadratically with the

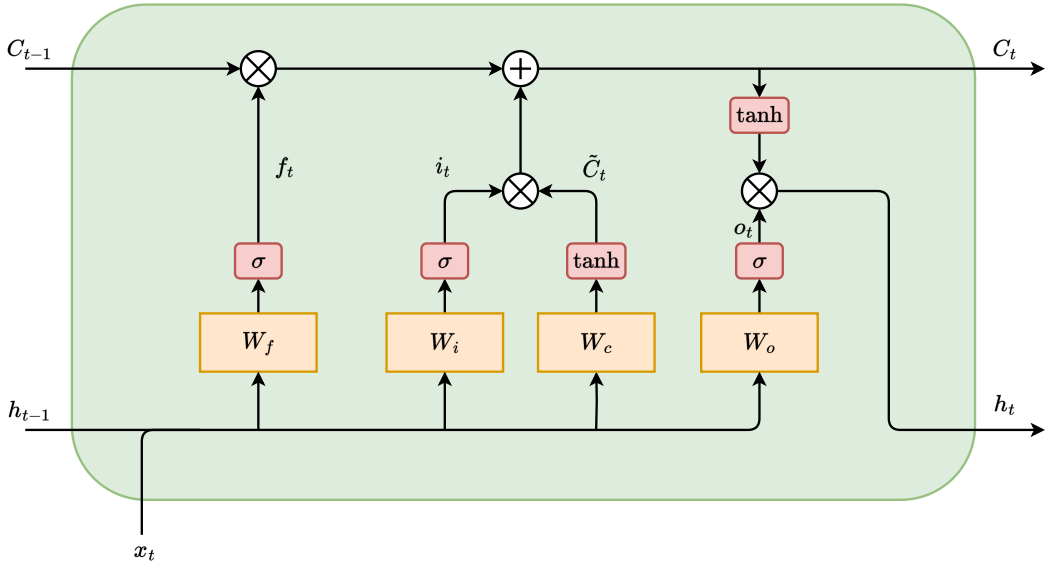


Figure 2.2: Internal structure of a single LSTM cell. Orange and red boxes represent learned linear projections and nonlinear activation functions, respectively. Circles inscribed with "x" and "+" represent element-wise multiplication and addition operations. Converging lines indicate vector concatenation.

sequence length [50], [55].

Vanilla RNN models use bounded activation functions such as the hyperbolic tangent or sigmoid functions to prevent exploding gradients during training. However, these activations remove useful gradient information at each time step and inhibit the model's ability to learn temporal relationships across long sequences [56]. Long short-term memory (LSTM) networks address this by introducing a direct path of information flow across a sequence [57]. Specifically, LSTMs maintain a cell state C_t and several gating mechanisms in addition to the standard RNN hidden state h_t. The cell state is essentially a scaled skip connection whose state is updated primarily through element-wise linear operations. The gating components allow the LSTM to be selective about the information that is retained across time steps. These changes ensure that there is a path for meaningful gradient information to flow without significant compression.

Figure 2.2 shows the internal structure of an LSTM layer. The internal update rules in the figure are defined as:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad \text{(input gate)}, \tag{2.4}$$

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad \text{(forget gate)}, \tag{2.5}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad \text{(cell candidate)}, \tag{2.6}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad \text{(cell state update)}, \tag{2.7}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad \text{(output gate), and} \tag{2.8}$$

$$h_t = o_t * \tanh(C_t) \quad \text{(hidden state update)}. \tag{2.9}$$

For each sequence step, the new cell state is a linear combination of the previous cell state C_t and new candidate cell value $\tilde{C}_t$. The amount of influence C_{t-1} exerts on C_t is dictated by the forget gate f_t, and the influence of $\tilde{C}_t$ on C_t depends on the input gate i_t. The output gate o_t scales the tanh-squashed cell state to produce the new hidden state h_t. Each of these architectural components helps the LSTM preserve long-term dependencies and avoid vanishing/exploding gradient issues when training using backpropagation through time (BPTT). One common alternative to the LSTM is the gated recurrent unit (GRU) from [58], which simplifies the gating mechanism to achieve similar performance with fewer parameters.

2.1.3 Relational Reasoning on Graph-Structured Data

Graphs provide an explicit model of relations between entities in a system. An entity is a discrete element that possesses its own properties/attributes and a relation is a set of properties that are shared between two entities. Mathemati-

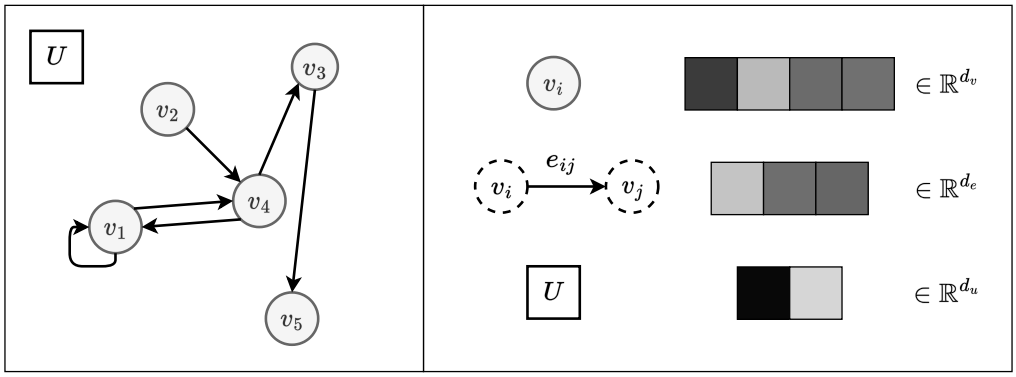
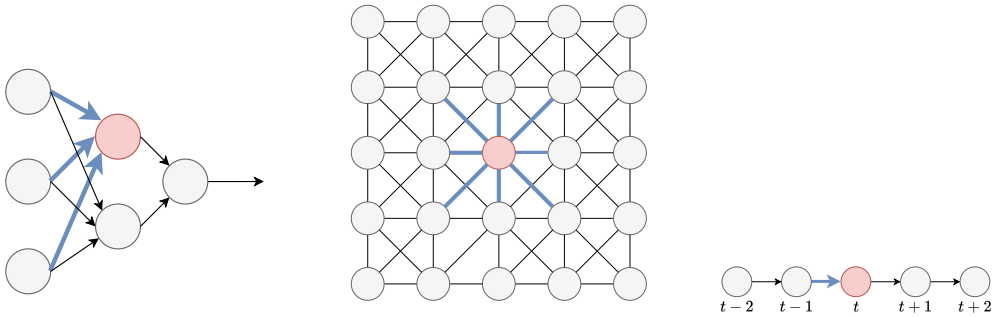


Figure 2.3: (Left) A simple example of a directed graph with global features. (Right) An attribute vector can be assigned to each individual node/edge/graph. In *homogeneous* graphs, all nodes and edges share a common feature space with all other nodes and edges, respectively. In *heterogeneous* graphs, the feature vectors of different node/edge classes need not share a vector space.

cally, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, U)$ is characterized by a set of nodes $\mathcal{V}$ which represent individual entities, a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ which describe relations between entities, and a set of global attributes $U \in \mathbb{R}^{d_u}$ which are associated to the entire graph instead of any individual node or edge (Figure 2.3).

Graphs can be directed or undirected: directed graph edges represent one-way relationships between a source and sink node (e.g., the edge between v_2 and v_4 in Figure 2.3). Edges in an undirected graph indicate a two-way connection shared between nodes (e.g., the edge(s) between v_1 and v_4). Each node $v \in \mathcal{V}$ and edge $e \in \mathcal{E}$ can also have associated feature vector of size d_v and d_e, respectively. For example, nodes in a chemical graph may represent individual atoms, edges may represent chemical bonds between atoms, and global features may include auxiliary information about the molecule itself. In this scenario each node may have a label feature that describes the type of atom and each edge may have a label describing the type of bond.

Standard deep learning building blocks also fit into the unified geometric learning framework and can be described as a graph operation [59]. A fully-



(a) Fully-connected layer (b) Convolutional layer (c) Recurrent layer

Figure 2.4: Graph representations of common machine learning building blocks.

connected (dense) layer is a graph where each neuron is a node with incoming edges from each node in the previous layer and outgoing edges to each node in the next layer (Figure 2.4a). Since all input neurons can interact to produce each output neuron value, the fully-connected layer imposes a very weak inductive bias and all structure must be learned from data. Every possible interaction between neurons has its own parameter value, so there is no parameter sharing and no invariance to transformations on the input data. Convolutional layers can be formulated as a graph operation where each pixel is a node connected to its nearest neighbors (Figure 2.4a). Since each node connects only to its local neighbors, convolutional layers impose a spatial inductive bias that limits the scope of interactions between nodes based on spatial proximity. Convolutional filters also enforce a translation invariance property by applying the same convolutional filters independently to each pixel. Finally, a recurrent layer is a Markov chain graph where each entity (sequence element) depends only on the previous output and current input. Similar to convolutional layers in the spatial domain, recurrent layers impose a temporal inductive bias. Recurrent layers are also time-shift invariant since they share parameters across time steps.

The inductive biases of the deep learning components described above impose

useful inductive biases for specific problem spaces (e.g., vision or sequence modeling). However, they are incapable of representing general relational structures that arise in graphs with more complex topologies. One major challenge is that nodes in a graph are presented as a set with no intrinsic ordering. To derive a semantic representation, a graph computation should therefore depend only on the relationships between objects and not their ordering. This property is called *permutation invariance*. Consider a function f which takes a set of vectors $\{x_1, \ldots, x_n\}$ as input and outputs a single vector:

$$f(\{x_1, \ldots, x_n\}) = \phi\left(\sum_{i=1}^{n} \psi(x_i)\right), \tag{2.10}$$

where $\psi(\cdot)$ is a function applied independently to each element and $\phi(\cdot)$ is a symmetric aggregation function (e.g., a sum or mean). If the aggregation function $\phi(\cdot)$ and the element-wise transformation $\psi(\cdot)$ do not depend on the ordering of the elements, Equation (2.10) will be permutation invariant [60].

While permutation invariance is a useful property for summarizing graphs, the symmetric aggregation step discards information about individual nodes and edges. Many graph processing techniques iteratively refine a latent representation of each node/edge while preserving identity information. These cases require graph computations that are *permutation equivariant*. For a permutation equivariant function, permuting the input elements produces a corresponding permutation of the output elements. Concretely, one can impose an ordering on the set $\{x_1, \ldots, x_n\}$ by stacking each element into the rows of a matrix $X = \begin{bmatrix} x_1 & \ldots & x_n \end{bmatrix}^T$. A function F is permutation equivariant if for some permutation matrix P we have:

$$F(PX) = PF(X). \tag{2.11}$$

For example, a linear transformation that is applied independently to each row of X is a permutation equivariant function.

It is also convenient to define the one-hop neighborhood of a node i as the set of all nodes that are connected to node i via a direct edge:

$$\mathcal{N}(i) = \{j : (i,j) \in \mathcal{E} \text{ or } (j,i) \in \mathcal{E}\}, \tag{2.12}$$

where (i,j) is an edge that originates from i and terminates in j if the graph is directed or an edge that is shared between nodes i and j if the graph is undirected. The degree of a node is defined as the number of neighbors in the node's neighborhood. For directed graphs, one can also define the in-degree and out-degree as the number of incoming and outgoing edges, respectively.

2.1.4 Graph Neural Networks

The mathematical formalisms defined in Section 2.1.3 provides a general recipe for designing graph neural networks (GNNs). Standard message-passing GNN layers consist of two main stages: an aggregation step and a combination step. The ℓ-th layer of a message-passing GNN updates node i's latent representation h_i as:

$$h_i^{(\ell)} = f\left(h_i^{(\ell-1)}, g\left(\left\{h_j^{(\ell-1)} : j \in \mathcal{N}(i)\right\}\right)\right), \tag{2.13}$$

where $g(\cdot)$ aggregates features from node i's neighbors into a single vector and $f(\cdot)$ combines the aggregated information with node i's previous state. Typically, g is a permutation invariant function, f is a permutation equivariant function, and the parameters of g and f are shared across nodes for each layer. Assuming g operates on the one-hop neighborhood of each node, sequentially applying L graph layers expands the node's receptive field to include information from nodes

up to L hops away. This composability allows GNNs to reason about both local and global relationships between nodes. Since the aggregation step is local to the node's neighborhood, the update step is independent of the graph size and can be applied in variable-sized graphs. The space and time complexity of a message-passing GNN is $\mathcal{O}(E)$ for a graph with E edges and N nodes, which is approximately $\mathcal{O}(N)$ for sparse graphs.

After graph processing, GNNs use a decoding/readout stage to convert latent node, edge, or graph features into a useful representation for the task at hand. Node-level tasks reason about individual nodes in the graph. For instance, a node-level task may classify the political affiliation of each user in a social network based on profile information. Edge-level tasks predict predict properties of individual edges (e.g., whether two users are friends in a social network). Finally, graph-level tasks attempt to make predictions on the entire graph. For example, a model may attempt to classify molecular graphs based on their chemical properties. The choice of decoding step is therefore an important design choice that must be chosen based on the task of interest.

2.1.5 Graph Network Architectures

This section defines several message-passing GNN layers that were explored in this work. Each layer follows the general form of Equation (2.13), differing primarily in their choice of f and g. The g function may be an isotropic aggregation function such as a sum or mean or it may be a more complex anisotropic function that adaptively weights the contributions of each neighbor. The f function is typically a single feed-forward dense layer or an MLP.

Perhaps the most common graph layer is the graph convolutional network (GCN) derived in [61]. The GCN update rule is a linearized approximation of

spectral graph convolution which updates the latent state of node i as:

$$h_i^{(\ell)} = \sigma\left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{\sqrt{\hat{d}_i \hat{d}_j}} h_j^{(\ell-1)} W^{(\ell)}\right), \tag{2.14}$$

where σ is a nonlinear activation function, $\hat{d}_i$ and $\hat{d}_j$ are the in-degrees of nodes i and j, respectively (including self-edges), and $W^{(\ell)} \in \mathbb{R}^{d' \times d}$ is a trainable weight matrix for the l-th layer.

As an alternative, the authors of [62] propose GraphSAGE, which use mean aggregation to compute the neighborhood features along with a learned residual connection in the combination step:

$$h_i^{(\ell)} = \sigma\left(h_i^{(\ell-1)} W_1^{(\ell)} + \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} h_j^{(\ell-1)} W_2^{(\ell)}\right). \tag{2.15}$$

Compared to the GCN layer, GraphSAGE requires twice as many trainable parameters for a fixed latent vector dimensionality. However, the learnable residual connection helps alleviate the oversmoothing problem where nodes become increasingly similar in deep GNNs [63].

Another common GNN layer is the graph isomorphism network (GIN) layer from [64]. Unlike GCN and GraphSAGE, GIN is a universal approximator of multi-sets and is provably as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test [65]. GIN updates each node's latent state as:

$$h_i^{(\ell)} = \text{MLP}^{(\ell)}\left((1 + \epsilon^{(\ell)}) \cdot h_i^{(\ell-1)} + \sum_{j \in \mathcal{N}(i)} h_j^{(\ell-1)}\right), \tag{2.16}$$

where the MLP is typically 2 or 3 layers and the ϵ parameter is a learnable bias term (often simply set to zero) applied to the skip connection. Many implemen-

tations also apply Batch Normalization [66] within the MLP.

Alternatively, the modified graph attention (GATv2) layer from [67], [68] is an example of an anisotropic GNN layer. The GATv2 update rule is given as follows:

$$h_i^{(\ell)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} \cdot W^{(\ell)} h_j\right), \tag{2.17}$$

where α_{ij} is a learned attention score that adaptively weights the importance of node j as:

$$\alpha_{ij} = \text{softmax}_j\left(e\left(h_i, h_j\right)\right) = \frac{\exp\left(e\left(h_i, h_j\right)\right)}{\sum_{j' \in \mathcal{N}(i)} \exp\left(e\left(h_i, h_j'\right)\right)}. \tag{2.18}$$

For GATv2, the edge weight $e\left(h_i, h_j\right)$ is a scalar that is computed by applying the following nonlinear transformation to each query-key attention pair:

$$e(h_i, h_j) = a^T \text{LeakyReLU}\left(W^{(\ell)} \cdot [h_i \parallel h_j]\right), \tag{2.19}$$

where $a \in \mathbb{R}^{2d'}$ is a trainable parameter vector and $\parallel$ is the vector concatenation operator. Note that Equation (2.17) does not include the previous latent state, so it is common for GATv2 implementations to include skip connections or self-edges in the graph itself. Compared to GCN, GATv2 requires approximately three times as many parameters for a fixed latent vector dimensionality. In many cases, the enhanced discrimination capabilities of GATv2 provide significant performance improvements compared to isotropic GNN layers like GCN or GraphSAGE.

The residual gated graph convolutional network (R-GGCN) from [69] is an even more expressive anisotropic GNN layer. The ℓ-th R-GGCN updates the

latent representation h_i for node i as:

$$h_i^{\ell+1} = h_i^{\ell} + \text{ReLU}\left(W_h^{\ell} h_i^{\ell} + \sum_{j \in \mathcal{N}(i)} \eta_{ij} \odot W_v^{\ell} h_j^{\ell}\right), \tag{2.20}$$

where $\odot$ is an element-wise Hadamard product. The η_{ij} component is a gating term that modulates the influence of node j on node i, defined as:

$$\eta_{ij} = \sigma\left(W_Q^{\ell} h_i^{\ell} + W_K^{\ell} h_j^{\ell} + W_e^{\ell} e_{ij}^{\ell}\right), \tag{2.21}$$

where $\sigma(\cdot)$ is the logistic sigmoid function, e_{ij}^{ℓ} is a feature vector for the edge connecting nodes i and j, and $W_h^{\ell}, W_e^{\ell}, W_Q^{\ell}, W_K^{\ell}, W_V^{\ell}$ are learnable weight matrices. Unlike for GATv2, η_{ij} is a vector of attention weights that independently modulates each dimension of the latent vector. To maintain a consistent scale across nodes with different neighborhood sizes, gate values are normalized such that $\sum_{j \in \mathcal{N}(i)} \eta_{ij} = 1$ for each node i. Some implementations of R-GGCN also propagate edge features across subsequent layers to improve performance in edge-level tasks [70]:

$$e_{ij}^{\ell+1} = e_{ij}^{\ell} + \text{ReLU}\left(W_Q^{\ell} h_i^{\ell} + W_K^{\ell} h_j^{\ell} + W_e^{\ell} e_{ij}^{\ell}\right). \tag{2.22}$$

2.2 Reinforcement Learning

Reinforcement learning (RL) systems train one or more agents to optimize their behavior based on feedback from their environment. This feedback comes in the form of a scalar reward signal that quantifies how well the agent achieves the desired objective. Unlike in supervised learning contexts where the learner receives ground truth labels for each observation, RL agents must explore their environment through trial-and-error to discover performant solutions to the task

at hand [71]. The feedback loop between the agent and its environment is often referred to as a perception-action cycle (PAC) in the autonomous sensing literature [11].

The RL problem is complicated by several factors. First, RL is a highly non-stationary problem. The distribution of scenarios the agent encounters often changes significantly over time as the agent explores the solution space and adapts its behavior. The reward signal is often sparse and may depend on the agent's behavior from many time steps in the past. For example, a chess-playing agent may only receive a reward at the end of each game which indicates whether it won, lost, or drew. Additional care must be taken when the environment state is stochastic and partially observable. However, RL agents can learn to solve complex, goal-oriented tasks with little human oversight when these challenges can be overcome.

RL has deep historical ties to the dynamic programming and optimal control literature [71]. Similar to dynamic programming, RL formulates the agent's interaction with its environment as a sequential decision making problem. When the agent observes the complete state of its environment, the RL problem is a Markov decision process (MDP) characterized by a 4-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ with the following components:

- State space $\mathcal{S}$: the set of all possible environment states.
- Action space $\mathcal{A}$: the set of all feasible actions the agent can take. The action space may vary over time and may depend on the environment state.
- Environment dynamics model $\mathcal{T}$: maps state-action pairs to a probability distribution over possible next states. For a state transition tuple (s, a, s') where s is the current state, a is the current action, and s' is the next state,

the dynamics model computes $\mathcal{T}(s, a, s') = p(S_t = s' | S_{t-1} = s, A_{t-1} = a)$, which is the probability of transitioning to state s' from state s after taking action a.

- Reward function $\mathcal{R}$: maps state-action pairs to an expected reward. The reward function has the form $R(s, a) = \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a]$, where R_t is the scalar reward at time t.

When the RL agent's perceptual abilities are limited or imperfect, the agent may not have direct access to the true environment state or transition dynamics. A *partially-observable* Markov decision process (POMDP) explicitly incorporates this ambiguity into the model by representing the problem as a 6-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$ [72]. The first four tuple elements match the MDP described above and the additional components are defined as follows:

- Observation space $\mathcal{O}$: the set of all possible *observations* available to the agent. These observations are typically noisy or incomplete representations of the true environment state.

- Observation dynamics function Ω: Maps state-action pairs to a probability distribution over possible next observations. The observation dynamics are defined as $\Omega(s, a, o') = p(O_t = o' | S_{t-1} = s, a_{t-1} = a)$ transition tuple (s, a, o').

Partial observability adds additional complexity to the RL problem. In addition to learning optimal control behavior, the agent must concurrently estimate the true state of its environment from its observation [73]. RL agents typically accomplish this by integrating several observations over time to form an internal belief state that approximates the true environment state. The environments

considered in this work are both partially observable and stochastic, so this dissertation uses "observation" and "state" interchangeably and uses s_t in place of o_t.

Given an environment and its (PO)MDP representation, RL agents typically learn a control policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ which can be represented as a probability distribution over possible actions for each environment state. The agent selects actions to maximize the cumulative reward over a time horizon of interest, known as the *return* Ψ_t. This work considers the infinite-horizon case which defines the return at time t as:

$$\Psi_t = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right], \tag{2.23}$$

where r_t is the reward from the environment and $\gamma \in [0, 1]$ is a discount factor that bounds the effective optimization horizon of the agent. Discount factors close to unity encourage the agent to maximize long-term rewards while smaller discount factors encourage myopic behavior that prioritizes immediate rewards.

Equation (2.23) depends only on the discount factor and the sequence of rewards $r_t, r_{t+1}, \ldots$ received from the environment at each time step. To derive an optimal policy which maximizes the return, the agent requires an optimization objective that is conditioned on the learned policy π. One such function is the state-value function $V(s)$, which gives the expected return when the agent begins in state s_t and follows policy π thereafter:

$$\begin{aligned} V_\pi(s_t) &= \mathbb{E}_\pi\left[\Psi_t | S_t = s_t\right] \\ &= \mathop{\mathbb{E}}_{a_{t:\infty} \sim \pi}\left[\sum_{i=0}^{\infty} \gamma^i R(s_{t+i}, a_{t+i}) \;\middle|\; S_t = s_t\right], \end{aligned} \tag{2.24}$$

where $a_{t:t+T}$ is notational shorthand for the action sequence $a_t, a_{t+1}, \ldots, a_{t+T}$.

Crucially, the value function obeys a recursive Bellman relation:

$$\begin{aligned} V_\pi(s_t) &= \mathop{\mathbb{E}}_{a_{t:\infty}\sim\pi}\left[\sum_{i=0}^{\infty}\gamma^i R(s_{t+i},a_{t+i}) \;\middle|\; S_t = s_t\right] \\ &= \mathop{\mathbb{E}}_{a_{t:\infty}\sim\pi}\left[R(s_t,a_t) + \sum_{i=1}^{\infty}\gamma^i R(s_{t+i},a_{t+i}) \;\middle|\; S_t = s_t\right] \\ &= \mathop{\mathbb{E}}_{a_{t:\infty}\sim\pi}\left[R(s_t,a_t) + \gamma\Psi_{t+1} \mid S_t = s_t\right] \\ &= R(s_t,a_t) + \gamma V_\pi(s_{t+1}). \end{aligned} \tag{2.25}$$

This recursion is the basis of temporal-difference (TD) learning methods which incrementally improve the value function estimate from individual state transitions [74].

Many RL algorithms instead choose to optimize the action-value function $Q_\pi(s,a)$, which is commonly referred to as the Q-function. The Q-function gives the expected return when the agent takes action a_t in state s_t and follows policy π for all subsequent time steps:

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_\pi\left[\Psi_t | s_t = s, a_t = a\right] \\ &= \mathop{\mathbb{E}}_{a_{t+1:\infty}\sim\pi}\left[\sum_{i=0}^{\infty}\gamma^i R(s_{t+i},a_{t+i}) \;\middle|\; s_t = s, a_t = a\right]. \end{aligned} \tag{2.26}$$

Like the standard value function, the Q-function can also be shown to satisfy a recursive Bellman relation when the Markov property holds.

Given Equations (2.24) and (2.26), an optimal policy $\pi^\star$ can be defined as a policy that maximizes the value or action-value functions for all states $s \in \mathcal{S}$. In many cases, the optimal policy is not unique and multiple policies may be equivalently optimal. These optimal policies share a corresponding optimal value function $V^\star(s)$ and Q-function $Q^\star(s,a)$. Since the policy and value functions are conditioned on one another, RL algorithms typically alternate between policy

and value function updates to iteratively improve both components.

Finally, the advantage function $A_\pi(s, a)$ describes the value of action a relative to the average action in state s under policy π. The definition of the advantage function follows directly from the value and Q-functions:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s). \tag{2.27}$$

The advantage function plays a central role in the policy gradient methods discussed in Section 2.2.1.

2.2.1 Proximal Policy Optimization

The proximal policy optimization (PPO) family of algorithms from [75] is the RL engine for the spectrum sharing portion of this work. PPO is a model-free policy gradient method that directly optimizes the parameters θ of a policy π_θ using gradient ascent. Traditional policy gradient techniques are *on-policy* algorithms that train the agent exclusively on data collected from the current policy without maintaining a replay buffer of previous experiences. PPO introduces a clipped optimization objective that prevents undesirably large updates to the policy parameters, enabling the agent to re-use past experiences and improve sample efficiency [75].

The classic policy gradient algorithm [76] optimizes the following objective at each time step t:

$$L^{\mathrm{PG}}(\theta) = \mathbb{E}_{\pi_\theta}\left[\hat{A}_t \log \pi_\theta(a_t|s_t)\right], \tag{2.28}$$

where $\hat{A}_t$ is an estimate of the advantage function from Equation (2.27), the $\log \pi_\theta(a_t|s_t)$ term denotes the log-probability of taking action a_t in state s_t under the current policy policy π_θ, and the expectation is taken over a batch of samples

collected under π_θ. Differentiating Equation (2.28) with respect to the policy parameters gives the vanilla policy gradient estimator:

$$\hat{g}^{\mathrm{PG}}(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t\right]. \tag{2.29}$$

The policy gradient in Equation (2.29) behaves as follows. If action a_t is superior to the average policy action in state s_t, the advantage $\hat{A}_t > 0$ and the gradient updates the policy parameters to *increase* the log-probability of a_t in state s_t. If $\hat{A}_t < 0$, the gradient update decreases the log-probability of a_t.

Equation (2.29) is conditioned on the current policy π_θ and produces destructively large gradients when applied to data collected under old policies. However, data re-use is critical for efficient learning with minimal environment interaction. To improve sample efficiency, PPO adopts a clipped policy objective that prevents destructive gradient updates when the training data was collected from a past policy. The PPO policy loss is defined as:

$$L_t^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right], \tag{2.30}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)}$ is the likelihood ratio between the current policy π_θ and the policy used to collect the training data $\pi_{\theta_{\mathrm{old}}}$. The clip operator is defined as:

$$\mathrm{clip}(x, a, b) = \begin{cases} a & \text{if } x \leq a, \\ x & \text{if } a < x < b, \\ b & \text{if } x > b. \end{cases} \tag{2.31}$$

Clipping the ratio to the range $[1-\epsilon, 1+\epsilon]$ prevents destructively large policy updates. The $\min(\cdot)$ operator makes the clipped objective a pessimistic lower

bound whose magnitude does not exceed that of the unclipped objective $r_t(\theta)\hat{A}_t$.

PPO trains a value function estimate V_θ using a squared error loss:

$$L_t^V(\theta) = \mathbb{E}\left[(V_\theta(s_t) - \bar{V}_t)^2\right], \tag{2.32}$$

where $\bar{V}_t$ is the discounted sum of rewards in the environment from time t to time $t+T-1$ for some number of time steps T. The PPO loss function also contains an entropy bonus that biases the optimization towards stochastic policies and encourage exploration. With these components, the complete PPO objective is defined as:

$$L_t^{PPO}(\theta) = \mathbb{E}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^V(\theta) + c_2\mathcal{H}(\pi_\theta(s_t))\right], \tag{2.33}$$

where $\mathcal{H}(\pi_\theta(s_t))$ is the entropy of the policy action distribution in state s_t and c_1, c_2 scale the value and entropy loss terms, respectively.

One major limitation of PPO is that it is highly sensitive to hyperparameter choices and often requires significant tuning across environments. This is because the value and entropy components in Equation (2.33) may have drastically different scales for each task. If the value and policy networks share parameters, c_1 must be chosen to ensure that the value loss does not dominate the policy loss or vice-versa. At the same time, c_2 must be sufficiently large to encourage exploration without causing the policy to become too random to converge to a performant solution. This sensitivity can be somewhat mitigated if the policy and value models do not share parameters. However, this increases the overall size of the model, slows training, and fails to eliminate hyperparameter sensitivity entirely [77].

Both vanilla policy gradient and PPO must estimate the advantage function

A_t from training data. PPO uses the generalized advantage estimator (GAE) from [78]. In its most general form, GAE estimates the advantage at time t as:

$$\hat{A}_t^{\text{GAE}} = \sum_{i=0}^{\infty} (\gamma \lambda^{GAE})^i \delta_{t+i}^V, \tag{2.34}$$

where γ is the discount factor, λ^{GAE} is a hyperparameter that controls the bias-variance trade-off of the estimator, and δ_t^V is a single-step temporal difference (TD) error defined as:

$$\delta_t^V = [r_t + \gamma V_\theta(s_{t+1})] - V_\theta(s_t). \tag{2.35}$$

Practical GAE implementations truncate the advantage estimate in Equation (2.34) to a finite sum over trajectories of T time steps and use the value model V_θ to compute the value of the final time step [75]:

$$\hat{A}_t^{\text{GAE, trunc}} = \delta_t + (\gamma \lambda^{GAE}) \delta_{t+1} + \cdots + (\gamma \lambda^{GAE})^{T-t+1} \delta_{T-1}. \tag{2.36}$$

Though PPO is considerably more stable and sample efficient than classic policy gradient techniques, it tends to underperforms state-of-the-art RL methods in terms of both learning efficiency and final performance. However, it is simple to implement and computationally inexpensive, making it a popular choice in environments that can quickly generate large quantities of training data. PPO is also used frequently in multi-agent systems [79] and large language model (LLM) fine-tuning [80].

2.2.2 Temporal-Difference Model Predictive Control

RL algorithms can be broadly divided into two classes: model-free or model-based. Model-free methods directly learn a policy and/or value function from environment interactions without explicitly learning a model of the environment's transition dynamics or reward function. The PPO algorithm discussed in Chapter 2.2.1 is a classic example of model-free deep RL. Since model-free methods do not require additional modeling components, they are often simple to implement and computationally less expensive than model-based approaches. However, they often lag behind their model-based counterparts in terms of sample efficiency and final performance [81].

Unlike the model-free case, model-based RL methods learn an explicit representation of the environment dynamics known as a *world model* [82]. World models can significantly improve learning efficiency by enabling the agent to generate synthetic training data [83], plan in the world model latent space [81], [83]–[85], or leverage rich model-based learning objectives to improve model-free representations [86]. Since world models necessarily capture the underlying structure of the environment, model-based agents can often generalize their behavior across related tasks or robotic embodiments [81]. The scalability and generalization capabilities of model-based RL open the door for foundational RL models which can be pre-trained on a large corpus of data and fine-tuned for specific tasks of interest.

For the search and track problem discussed in Chapter 4, the RL agents are trained using the temporal-difference model predictive control (TD-MPC) technique [81], [84]. This section describes the TD-MPC2 algorithm introduced in [81], which augments the original formulation from [84] with several architectural updates to improve its generalization and scalability. TD-MPC2 explicitly

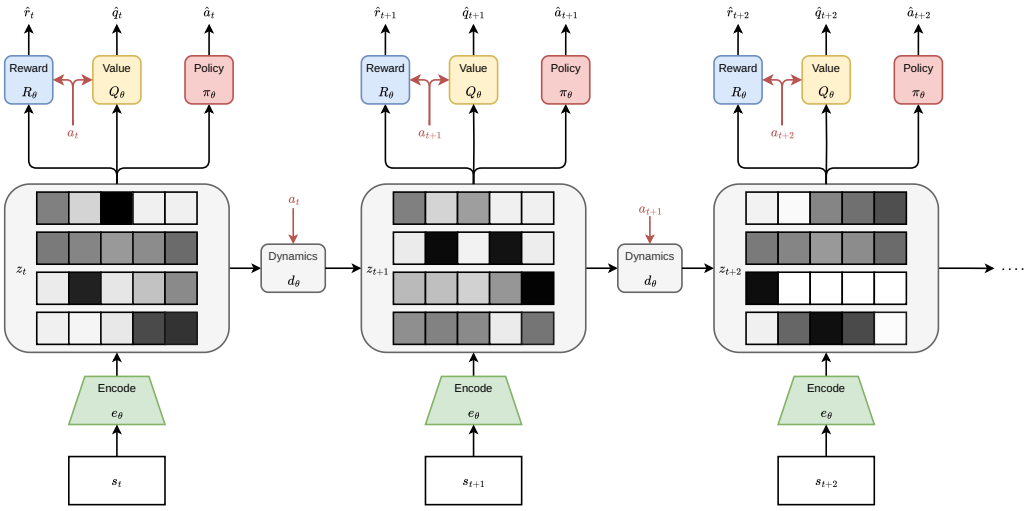


Figure 2.5: TD-MPC2 world model components (Figure from [81]). At each time step, TD-MPC2 encodes the observed state into a sparse latent vector and predicts future rewards, values, and state transitions. TD-MPC2 does not use a decoder and performs all prediction and planning in the latent space.

learns the environment's transition and reward dynamics (i.e., a world model), then performs model predictive control (MPC) planning in the world model to select performant actions. TD-MPC2 achieves state-of-the-art performance on a wide variety of continuous control benchmarks without task-specific hyperparameter tuning, making it a strong choice for rapidly prototyping the sensor control agents in this work. To further accelerate training, this work provides a highly performant Jax implementation of the algorithm.[1]

The TD-MPC2 world model consists of five sub-components:

- Encoder e_θ: Maps raw environment observations s_t to a latent vector z_t in the world model latent space.

- Dynamics model $d_\theta(z_t, a_t)$: Predicts the next latent state z_{t+1} from the current latent state z_t and action a_t.

[1] A performant Jax implementation of TD-MPC2 can be found at https://github.com/ShaneFlandermeyer/tdmpc2-jax

- Reward model $R_\theta(z_t, a_t)$: Predicts the single-step reward r_t from the current latent state z_t and action a_t.
- Value model $Q_\theta(z_t, a_t)$: Predicts the action-value function from the current latent state z_t and action a_t.
- Policy $\pi_\theta(z_t)$: Computes an action distribution for action a_t from the current latent state z_t. For continuous action spaces, the policy network outputs a multivariate Gaussian distribution with learned parameters.

All world model components aside from the encoder are implemented as multilayer perceptron (MLP) networks that operate on latent vectors from the encoder and dynamics model. The hidden layers in each MLP use LayerNorm and Mish activations [87]. The encoder architecture can be selected based on the modality of the environment observations. For instance, an MLP encoder is suitable for vector-valued observations while a convolutional neural network (CNN) encoder is more appropriate for image-based observations. The search and track component of this work operates on graph-structured observations, so it uses a graph neural network (GNN) encoder as described in Chapter 4. The world model also incorporates several recent techniques to improve stability and scalability, including distributional reward/value losses [88], Q-network ensembles [89], [90], and Layer Normalization [51].

To prevent exploding gradients and bias the latent space towards sparsity, TD-MPC2 applies a SimNorm activation to the encoder and dynamics model outputs [91]. SimNorm projects the latent state vector onto L simplices of dimension V using a softmax operation. Specifically, given a latent vector $z \in \mathbb{R}^{L \times V}$, SimNorm partitions z into L V-dimensional simplices $g_1, \ldots, g_L$ and forms the

normalized latent vector as:

$$\text{SimNorm}(z) = [g_1,\ \ldots,\ g_L], \quad g_i = \frac{e^{z[i\cdot V:i\cdot V+V]/\tau}}{\sum_{j=1}^{V} e^{z[i\cdot V+j]/\tau}}, \tag{2.37}$$

where $z[i : i+N]$ extracts elements i through $i+N-1$ for the vector z and τ is a temperature parameter that controls the degree of sparsity in the normalized representation. Smaller values of τ drive the latent vectors towards one-hot encodings, while larger values produce a more uniform distribution across each simplex.

TD-MPC2 iteratively updates the world model parameters θ to optimize the following joint objective function over an H-step trajectory:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\sum_{t=0}^{H-1} \lambda^t \left(\mathcal{L}_{\text{dyn}}(\theta) + \mathcal{L}_{\text{R}}(\theta) + \mathcal{L}_Q(\theta)\right)\right], \tag{2.38}$$

where $\lambda \in (0, 1]$ is a temporal weighting factor. The $\mathcal{L}_{\text{dyn}}$ term constrains the latent dynamics model to produce outputs that are temporally consistent with the encoder:

$$\mathcal{L}_{\text{dyn}}(\theta) = \left\| d_\theta(z_t) - \texttt{sg}(e_\theta(s_{t+1})) \right\|^2, \tag{2.39}$$

where $\texttt{sg}(\cdot)$ is the stop-gradient operator.

The second term in the world model objective is a reward prediction loss defined as:

$$\mathcal{L}_R(\theta) = CE(\hat{r}_t, r_t), \tag{2.40}$$

where $CE(\cdot)$ is a categorical cross-entropy loss, r_t is the reward from the environment at time t, and $\hat{r}_t$ is the reward prediction from the world model. The final term in Equation (2.38) is a value prediction loss whose form is identical to

the reward prediction objective above:

$$\mathcal{L}_Q(\theta) = CE(\hat{q}_t, q_t), \tag{2.41}$$

where $\hat{q}_t$ is the value estimate from the world model, $q_t = r_t + \bar{Q}(z_{t+1}, p(z_{t+1}))$ is a one-step temporal-difference (TD) target, and $\bar{Q}$ is a target Q-network whose parameters are the exponentially moving average of the parameters of Q_θ. Note that the loss terms in Equations (2.40) and (2.41) are both formulated as a discrete classification problem with a cross-entropy loss, which has been shown to improve robustness and scalability compared to the regression-style losses that are typical in traditional deep RL algorithms [88], [92].

The TD-MPC2 agent also learns a policy prior π_θ that guides the planning procedure in the world model latent space. The policy is trained in parallel with the world model components using the maximum-entropy soft actor-critic (SAC) loss similar to [93]:

$$\mathcal{L}_\pi(\theta) = \mathbb{E}\left[\sum_{t=0}^{H} \lambda^t \left[\alpha Q(z_t, \pi(z_t)) - \beta \mathcal{H}(\pi(z_t))\right]\right], \tag{2.42}$$

where $\mathcal{H}(\cdot)$ is the entropy of the policy action distribution. To decouple the scale of $\mathcal{L}_\pi$ from the scale of Q, the authors dynamically adapt α so that the first term lies nominally in the range $[0, 1]$ and set β to a small constant value.

Traditional MPC controllers simulate candidate trajectories over an H-step time horizon and select the action sequence that maximizes the expected cumulative reward:

$$\arg\max_{a_{t:t+H-1}} \mathbb{E}\left[\sum_{i=0}^{H-1} \gamma^i R(s_{t+i}, a_{t+i})\right]. \tag{2.43}$$

To identify good control sequences, MPC must simulate a large population of can-

didate trajectories in the environment simulator. For sensor models with even moderately complex dynamics, this introduces significant runtime computation that may preclude the use of MPC in resource-limited systems or systems with strict real-time operational requirements. Consequently, MPC planning horizons are typically short and Equation (2.43) may not produce globally optimal behaviors beyond the planning horizon.

TD-MPC2 addresses these shortcomings in two ways. First, TD-MPC2 performs all planning simulation using trajectories sampled from the world model rather than the original environment simulator. Since each world model component is a simple MLP network, the planner can sample hundreds of candidate trajectories in parallel using hardware accelerators such as GPUs or TPUs. This reduces runtime computational requirements by several orders of magnitude compared to CPU-based environment simulators. Recent work in the literature has also approximated the TD-MPC2 planning procedure itself using a neural network trained via imitation learning, further reducing inference-time latency [85] and improving sample efficiency.

In addition, TD-MPC2 incorporates a learned value estimate Q into the planning objective. This enables the agent to optimize its policy over long time horizons without increasing the MPC planning horizon H. The TD-MPC2 planner thus selects its action distribution to maximize the following objective:

$$\arg \max_{a_{t:t+H} \sim \mathcal{N}(\mu, \sigma^2)} \mathbb{E}\left[\sum_{i=0}^{H-1} \gamma^i R_\theta(z_{t+i}, a_{t+i}) + \gamma^H Q(z_{t+H}, a_{t+H})\right], \tag{2.44}$$

where $\mu = \mu_{t:t+H} \in \mathbb{R}^{H \times N_A}$ and $\sigma = \sigma_{t:t+H} \in \mathbb{R}^{H \times N_A}$ contain the N_A-dimensional mean vectors and covariance diagonals for the action distribution at each step in the planning horizon.

Algorithm 1 TD-MPPI Planning

Require: World model components: $(e_\theta, d_\theta, R_\theta, Q_\theta, \pi_\theta)$
Plan horizon: H
Initial state: s_0
Initial plan distribution: $\mu^0_{0:H-1}, \sigma^0_{0:H-1}$
Number of policy prior samples: N_π
MPPI population size: N

$z_0 \leftarrow e_\theta(s_0)$ ▷ *Encode initial state*

▷ *Sample trajectories from policy prior*

for $i = 1, \ldots, N_\pi$ **do**
  for $t = 0, \ldots, H-1$ **do**
    $a^i_t \sim \pi_\theta(z^i_t)$
    $z^i_{t+1} \leftarrow d_\theta(z^i_t, a^i_t)$
  $v^i \leftarrow \texttt{estimate_value}(z_0, a^i_{0:H-1})$

▷ *MPPI planning*

for $j = 0, \ldots, J-1$ **do**
  for $i = N_\pi, \ldots, N$ **do**
    $a^i_{0:H-1} \sim \mathcal{N}(\mu^j_{0:H-1}, \sigma^j_{0:H-1})$
    $v^i \leftarrow \texttt{estimate_value}(z_0, a^i_{0:H-1})$

  ▷ *Update plan distribution*

  Select k best trajectories $\Gamma^\star_{1:k}$ and their values $v^\star_{1:k}$
  Compute trajectory scores: $\phi^\star_{1:k} \leftarrow \mathrm{softmax}(\tau v^\star_{1:k})$
  $\mu^{j+1}_{0:H-1} \leftarrow \sum_{i=1}^k \phi^\star_i \cdot \Gamma^\star_i$
  $\sigma^{j+1}_{0:H-1} \leftarrow \sqrt{\sum_{i=1}^k \phi^\star_i \cdot \left(\Gamma^\star_i - \mu^{j+1}_i\right)^2}$

return Final plan distribution $(\mu^J_{0:H-1}, \sigma^J_{0:H-1})$

TD-MPC2 uses the model predictive path integral (MPPI) control algorithm from [94] to derive action distributions that solve Equation (2.44). MPPI is a sampling-based control algorithm that iteratively refines an action distribution based on a large population of candidate solutions. In this work, the MPPI planner updates the parameters of a diagonal multivariate Gaussian distribution based on trajectories sampled from the world model. To disambiguate between the general MPPI algorithm and the TD-MPC2 planning procedure, this section refers to the latter as TD-MPPI.

TD-MPPI begins by embedding the current environment state s_0 into the world model to produce an initial latent state $z_0 = e_\theta(s_0)$. Next, the planner initializes an action distribution $(\mu^0_{0:H-1}, \sigma^0_{0:H-1})$ for each time step, where the standard deviation values in $\sigma^0_{0:H-1}$ are set to a constant large value to encourage exploration. At each optimization step j, the planner samples N independent H-step trajectories that start in the state z_0 and sample action from the current plan distribution. Each trajectory is evaluated according to Equation (2.44), where the reward and value estimates are computed using the world model components R_θ and Q_θ. TD-MPPI extracts the top-k trajectories from the overall population and updates the plan distribution for the next iteration as:

$$\mu^{j+1} = \frac{\sum_{i=1}^{k} \Omega_i \Gamma_i^\star}{\sum_{i=1}^{k} \Omega_i}, \;\; \sigma^{j+1} = \sqrt{\frac{\sum_{i=1}^{k} \Omega_i \left(\Gamma_i^\star - \mu^{j+1}\right)^2}{\sum_{i=1}^{k} \Omega_i}}, \tag{2.45}$$

where $\Gamma_i^\star$ is the i-th trajectory in the top-k, $\Omega_i = \exp\left(\tau v^\star_{\Gamma,i}\right)$ is a softmax weighting factor, and τ is a temperature parameter which controls how aggressively the planner converges on high-value solutions.

In large action spaces, the procedure described above may require many iterations and a large population size to converge to a performant solution. To

accelerate planning, TD-MPC2 also incorporates the learned policy π_θ to reduce the search space during optimization. Specifically, TD-MPPI reserves $N_\pi \ll N$ members of the population as policy prior samples whose actions are sampled directly from the policy network rather than the plan distribution. Since the policy is trained on Equation (2.42) to maximize the long-term value, the policy prior provides a set of high-value initial solutions that guide the optimization process. The policy network also samples the final action a_H used to compute the terminal value estimate in Equation (2.44).

Algorithm 2 Model-based Value Estimation

Require: World model components: $(e_\theta, d_\theta, R_\theta, Q_\theta, \pi_\theta)$
Plan horizon: H
Initial latent state: z_0
Action sequence: $a_{0:H-1}$

▷ *Evaluate actions in the world model*

$v \leftarrow 0$
for $t = 0, \ldots, H-1$ **do**
  $r_t \leftarrow R_\theta(z_t, a_t)$
  $z_{t+1} \leftarrow d_\theta(z_t, a_t)$
  $v \leftarrow v + \gamma^t r_t$

▷ *Compute terminal value estimate*

$a_H \sim \pi_\theta(z_H)$
$v \leftarrow v + \gamma^H Q_\theta(z_H, a_H)$
return v

After executing the TD-MPPI planning procedure for J iterations, the agent samples its action from the first time step of the final plan distribution (μ_0^J, σ_0^J). Rather than discarding the remaining plan after each decision step, the agent "warm-starts" each step by setting the initial mean μ^0 equal to the mean of the previous plan, shifted forward by one time step. To prevent premature convergence, the standard deviation σ^0 is initialized to a large value and the solution from the previous time step is not re-used. Algorithm 1 summarizes the com-

plete TD-MPPI planning procedure. Note that the trajectory sampling and value estimation in Algorithm 1 are performed in parallel for all members of the population. Algorithm 2 describes the subroutine for estimating the value of a candidate trajectory in the world model. The value estimation step requires H forward passes through the dynamics and reward model and a single pass through the policy and value networks for each trajectory.

2.3 Single-Object Tracking

This section outlines the problem of tracking a single object in the presence of measurement noise, clutter, and missed detections. Since exactly one object is present throughout the scenario, the tracker does not need to adaptively infer the number of objects or object birth/death. This simplifies the tracking problem considerably since sensor measurements only originate from the lone tracked object or from clutter. Though the contributions of this dissertation focus on the multi-object case, single-object tracking provides a useful context for introducing kinematic/measurement models and Bayesian filtering techniques such as the Kalman filter. Moreover, many multi-object trackers make assumptions that decompose much of the multi-object tracking problem into independent single-object sub-problems.

Transition Models

The tracking literature typically models the kinematic state of each object as a Markov chain whose distribution evolves as:

$$f(x_k|x_{k-1}) = p_k(x_k|x_{k-1}), \tag{2.46}$$

where p_k is a probability density known as the transition kernel [95]. One common choice of transition kernel adds zero-mean Gaussian noise q_{k-1} to a deterministic state transition function $f(\cdot)$. For this case, the object state evolves from time step $k-1$ to k as:

$$x_k = f_{k-1}(x_{k-1}) + q_{k-1}, \quad q_{k-1} \sim \mathcal{N}(0, Q_{k-1}), \tag{2.47}$$

where Q_{k-1} is a noise covariance matrix that captures uncertainty in the state estimate. Since $f(\cdot)$ is deterministic and the noise is zero-mean, the transition kernel is a multivariate Gaussian of the form

$$p_k(x_k|x_{k-1}) = \mathcal{N}(x_k; f_{k-1}(x_{k-1}), Q_{k-1}). \tag{2.48}$$

For example, a continuous white noise acceleration (CWNA) model assumes that the object's velocity is constant and its acceleration is zero-mean white Gaussian noise in each dimension. For 2-D motion, the CWNA model updates the object state in each dimension $x = [x_{\text{pos}},\ x_{\text{vel}}]^T$ as:

$$x_k = F_k x_{k-1} + q_{k-1}, \quad q_{k-1} \sim \mathcal{N}(0, Q_{k-1}) \tag{2.49}$$

with

$$F_{CV} = I_2 \otimes \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, \text{ and } Q_{CV} = q \begin{bmatrix} \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^2}{2} & \Delta t \end{bmatrix}, \tag{2.50}$$

where Δt is the time difference between state transitions and q is the process noise intensity. The dynamics in Equation (2.50) are a linear function of the input state. Special care must be taken during state estimation when the motion profiles include complexities such as nonlinearities or abrupt maneuvers.

Measurement Models

In addition to a transition dynamics model, tracking algorithms also require statistical models for the sensor(s) of interest. These sensor models describe a mapping from object states to measurements and characterize uncertainties in the measurement and detection processes. Many models also describe processes such as false alarms due to thermal noise or clutter measurements from the sensing environment. Even in the single-object case, these clutter processes introduce data association ambiguity that must be resolved in the tracker. However, this section focuses on the case when associations are known a priori. Further discussion of data association techniques is reserved for Section 2.4.

Let $p_d(x_k)$ denote the probability that an object with state x_k is detected at time k by the sensor of interest. If the object is detected, the sensor produces a measurement z_k which is distributed according to a likelihood function of the form:

$$g_k(z_k|x_k) = f(z_k|x_k). \tag{2.51}$$

As for the transition dynamics, it is common to model each measurement as deterministic function of the true object state $h(\cdot)$ that is corrupted by Gaussian noise:

$$z_k = h_k(x_k) + v_k, \quad v_k \sim \mathcal{N}(0, R_k), \tag{2.52}$$

where R_k is a covariance matrix that quantifies uncertainties or noise in the measurement process. This results in a Gaussian-distributed likelihood function of the form:

$$g_k(z_k|x_k) = \mathcal{N}(z_k; h_k(x_k), R_k). \tag{2.53}$$

With probability $1 - p_d(x_k)$, the object is not detected and produces no mea-

surement at the sensor. Define Z_k as a measurement set that contains the measurement vector if the object is detected and is empty otherwise. The detection-conditioned likelihood function for this set has the form:

$$f(Z_k|x_k) = \begin{cases} 1 - p_d(x_k) & \text{if } Z_k = \emptyset, \\ p_d(x_k) \cdot g_k(z_k|x_k) & \text{if } Z_k = \{z_k\}. \end{cases} \tag{2.54}$$

Using the nomenclature of Section 2.4.1, the measurement set Z_k is a Bernoulli random finite set with existence probability $p_d(x_k)$ and state density $g_k(z_k|x_k)$. It is also standard to assume that detections and measurements are i.i.d. for a given object state.

The linear-Gaussian model is perhaps the simplest type of measurement model:

$$z_k = H_k x_k + v_k, \quad v_k \sim \mathcal{N}(0, R_k), \tag{2.55}$$

where the $H_k \in \mathbb{R}^{n_z \times n_x}$ for some measurement dimensionality n_z and state dimension n_x. In Equation (2.55), each measurement dimension is a noise-corrupted linear combination of the input state, and state dimensions are not measured if their corresponding column in H_k is a zero vector.

This work also explores several scenarios where the measurement model is nonlinear. A Nonlinear Gaussian model has the form below:

$$z_k = h(x_k) + v_k, \quad v_k \sim \mathcal{N}(0, R_k), \tag{2.56}$$

where $h(x_k)$ is an arbitrary nonlinear function of the object state. For example, a radar or lidar sensor may extract range and bearing measurements $z_k = [r,\ \theta]^T$ from an object's 2D position (p_x, p_y). This range-bearing model is characterized

by:

$$h(x_k) = \begin{bmatrix} \sqrt{p_x^2 + p_y^2} \\ \arctan\left(p_y/p_x\right) \end{bmatrix} + \mathcal{N}(0, R_k), \quad R_k = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\theta^2 \end{bmatrix}, \tag{2.57}$$

where σ_r^2 and σ_θ^2 are (independent) error variances in the range and bearing dimensions, respectively. As for transition models, introducing nonlinearity generally requires a more complex state estimation filter.

2.3.1 Tracking with Known Associations

This section presents the standard Bayesian filtering model in the special case when data associations between measurements and the object are known and fixed [95]. Bayesian filters maintain a probability density $f(x_k|z_{1:k})$ that estimates the state x_k of the object at time k, conditioned on the measurement history $z_{1:k} = \{z_1, \ldots, z_k\}$. Bayesian filtering algorithms generally decompose the problem into alternating prediction and update steps, producing a new state density when new measurements arrive at the sensor. Though many Bayesian filtering techniques exist, this work focuses primarily on the Kalman filter and its variants.

The prediction step of a Bayesian filter propagates the object state to the next time step using the Chapman-Kolmogorov equation [95]:

$$f(x_k|z_{1:k-1}) = \int f_k(x_k|x_{k-1}) f(x_{k-1}|z_{1:k-1}) dx_{k-1}, \tag{2.58}$$

where $f_k(x_k|x_{k-1})$ is a Markov transition kernel, $z_{1:k-1}$ is the sequence of measurements up to and including time $k-1$, and $f(x_{k-1}|z_{1:k-1})$ is the state density from the previous time step. Given $f(x_{k-1}|z_{1:k-1})$, Equation (2.58) can be computed without knowledge of past or current measurements. The predicted state

is often referred to as the Bayesian *prior* state density since it depends only on information from previous time steps.

After prediction, the measurement update step incorporates new measurements into the state density using Bayes rule:

$$f(x_k|z_{1:k}) = \frac{g(z_k|x_k)f(x_k|z_{1:k-1})}{f(z_{1:k})} \propto g(z_k|x_k)f(x_k|z_{1:k-1}), \tag{2.59}$$

where $g(z_k|x_k)$ is the measurement likelihood function defined in Equation (2.53). Since the Bayesian prior $f(x_k|z_{1:k-1})$ summarizes all past measurement information, Equation (2.59) only evaluates the likelihood function for the current measurement z_k. The resulting density $f(x_k|z_{1:k})$ is known as the *posterior* density for time k. Given a posterior density, one can extract a final state estimate of for the object of interest using a minimum mean square error (MMSE) or maximum a posteriori (MAP) estimator [96]. Equation (2.59) implicitly assumes that sensor detects the object at each time step. If the object is missed and measurement z_k is not received, the posterior density is equal to the prior density after prediction.

2.3.2 Kalman Filtering

Evaluating Equations (2.58) and (2.59) in closed form may be difficult for arbitrary transition/measurement models. However, the prediction and update steps reduce to a simple form when the underlying distributions are Gaussian and the models are linear [97]. The linear Kalman filter assumes the object state density $f(x_k|z_{1:k})$ is a multivariate Gaussian with mean vector $\bar{x}_k \in \mathbb{R}^{n_x}$ and covariance $P_k \in \mathbb{R}^{n_x \times n_x}$, and that the transition/measurement models are linear functions of the form of Equations (2.49) and (2.55), respectively [98]. The remainder of

this section follows the standard tracking notation where the subscript $(\cdot)_{k|k-1}$ represents a quantity predicted from time $k-1$ to time k and $(\cdot)_{k|k}$ indicates a quantity after the measurement update step at time k.

Assuming the posterior density from time $k-1$ is a Gaussian density of the form $f_{k-1|k-1} = \mathcal{N}(\bar{x}_{k-1|k-1}, P_{k-1|k-1})$, the Kalman prediction step produces a new Gaussian distribution with parameters:

$$\bar{x}_{k|k-1} = F\bar{x}_{k-1|k-1}, \text{ and} \tag{2.60}$$

$$P_{k|k-1} = FP_{k-1|k-1}F^T + Q_{k-1}, \tag{2.61}$$

where F is the state transition function in matrix form and Q_{k-1} is the process noise covariance for the state transition model. In other words, the predicted mean is simply the previous mean propagated through the linear transition model. The predicted covariance transforms the previous covariance through the model and adds process noise to account for additional uncertainty in the model itself.

The Kalman update step incorporates new measurement information from the current time step into the state density. Given a measurement $z_k \in \mathbb{R}^{n_z}$, the residual $y_k \in \mathbb{R}^{n_z}$ is defined as the difference between the predicted state and the measurement at time k:

$$y_k = z_k - Hx_{k|k-1}. \tag{2.62}$$

Next, define the Kalman gain matrix $K \in \mathbb{R}^{n_x \times n_z}$ as:

$$K = \bar{P}_{k|k-1}H^T \left(HP_{k|k-1}H_k^T + R_k\right)^{-1} = \bar{P}_{k|k-1}H^T S_k^{-1}. \tag{2.63}$$

Intuitively, the Kalman gain can be interpreted as a ratio between the confidence

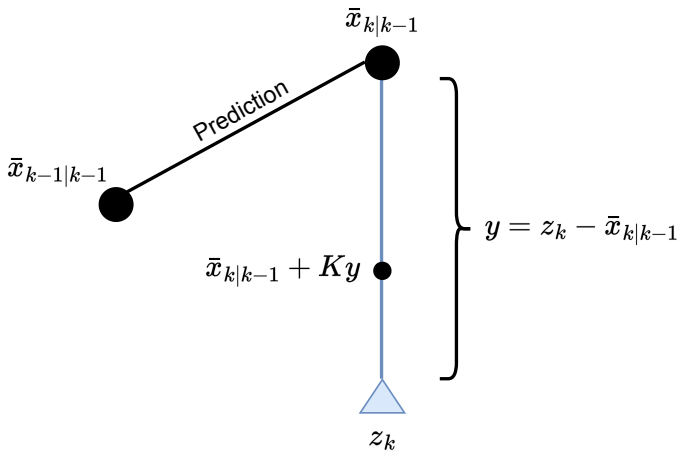


Figure 2.6: Illustration of the Kalman update for a two-dimensional state/measurement space. The posterior mean is a convex combination of the measurement and prediction, where its location along the line is determined by the Kalman gain.

in the filter prediction compared to the confidence in the new measurement. Kalman gain values near zero indicate higher confidence in the prediction, while values near unity indicate higher confidence in the measurement. The matrix $S = \left(HP_{k|k-1}H_k^T + R_k\right)^{-1} \in \mathbb{R}^{n_z \times n_z}$ is known as the innovation covariance and describes the total uncertainty due to both the measurement and prediction processes in measurement space.

Given the residual and Kalman gain, the Kalman filter applies the Bayesian update rule from Equation (2.59) to the predicted density to produce a Gaussian posterior distribution with parameters:

$$x_{k|k} = \bar{x}_{k|k-1} + Ky_k, \text{ and} \tag{2.64}$$

$$P_{k|k} = (I - KH)P_{k|k-1}, \tag{2.65}$$

where I is an identity matrix. Figure 2.6 shows the intuition behind the update step. The mean is a linear combination of the prediction and measurement, where

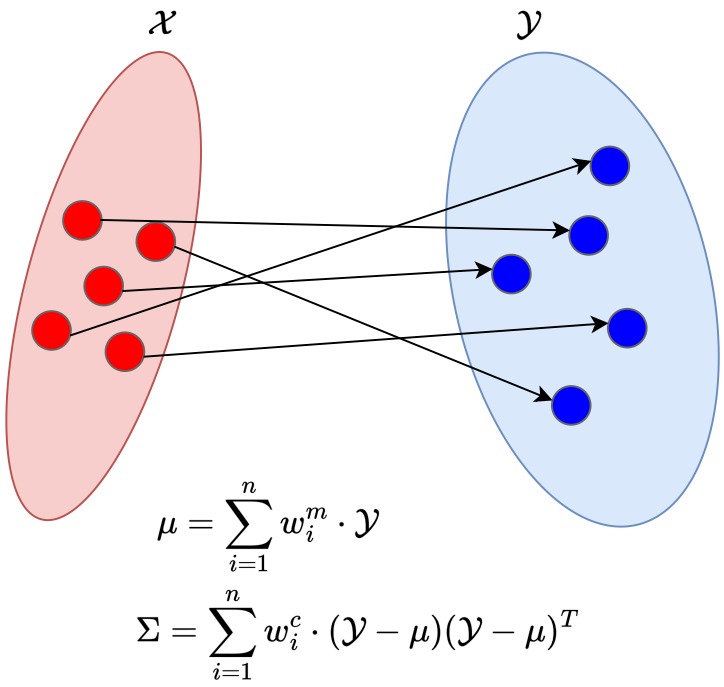


Figure 2.7: The unscented transform and its equations

a large Kalman gain biases the result towards the measurement and a small gain biases the estimate towards the prediction. The Kalman gain also reduces the covariance proportionally due to the term in parentheses in Equation (2.65).

Since the input density $f_{k-1|k-1}$ and output density $f_{k|k}$ are multivariate Gaussian distributions, the Gaussian distribution is known as a *conjugate prior* density for the Kalman filter. Conjugacy is a highly desirable property for state estimation. Since the density does not change form after the prediction and update, Equations (2.60)-(2.65) can be applied identically at each time step.

The standard Kalman filter defined in Equations (2.60)-(2.65) is optimal in an MMSE sense when the transition and measurement models are linear and Gaussian [98]. However, many practical applications require nonlinear models or more complex representations of the state density (e.g., due to occluded regions

of the state space). The extended Kalman filter (EKF) and unscented Kalman filter (UKF) are two such variants that accommodate nonlinear models [99], [100]. Though the EKF finds significant use in many contexts, the UKF generally outperforms it for models with modest or severe nonlinear behavior [101]. Additionally, the EKF requires Jacobian matrices which must be derived by hand for each model of interest or estimated using numerical methods. For these reasons, the remainder of this section focuses exclusively on the UKF and all single-object state estimation is performed using a UKF.

Like the linear Kalman filter, the UKF applies alternating predict and update steps to estimate the mean and covariance of the system state density. Rather than computing these quantities with a closed-form expression, the UKF applies the unscented transform to estimate the distribution parameters. The unscented transform uses a small set of query values known as sigma points to approximate the statistics of a probability distribution transformed by some function (Figure 2.7). Given a set of sigma points $\mathcal{X}$ and an arbitrary function f, the unscented transform first applies the function to each point to produce a new set $\mathcal{Y}$, where:

$$\mathcal{Y} = f(\mathcal{X}). \tag{2.66}$$

The unscented transform also requires a mean weight w_i^m and covariance weight w_i^c for each sigma point. Given these weights, the statistics of the transformed distribution can be estimated from a weighted sample mean and covariance:

$$\bar{y} = \sum_{i=1}^{n} w_i^m \mathcal{Y}_i, \text{ and} \tag{2.67}$$

$$P_y = \sum_{i=1}^{n} w_i^c (\mathcal{Y}_i - \bar{y})(\mathcal{Y}_i - \bar{y})^T, \tag{2.68}$$

where $\bar{y}$ and P_y are the mean vector and covariance of the transformed distribution, respectively.

Many techniques and heuristics have been developed to select sigma points and their weights. For an n-dimensional state vector, the method in [101] computes $2n+1$ sigma points according to:

$$\mathcal{X}_i = \begin{cases} \mu & \text{if } i = 0, \\ \mu + \left[\sqrt{(n+\lambda)\Sigma}\right]_i & \text{if } i = 1, \ldots, n, \\ \mu - \left[\sqrt{(n+\lambda)\Sigma}\right]_{i-n} & \text{if } i = n+1, \ldots, 2n, \end{cases} \tag{2.69}$$

where the $[\cdot]_i$ operator extracts the i-th row of the input matrix and the square root operation is a matrix square root (e.g., a Cholesky decomposition [102]). The λ term follows from user-specified parameters α and κ as:

$$\lambda = \alpha^2(n+\kappa) - n. \tag{2.70}$$

The mean weight for sigma point i is given by:

$$w_i^m = \begin{cases} \frac{\lambda}{n+\lambda} & \text{if } i = 0, \\ \frac{1}{2(n+\lambda)} & \text{if } i = 1, \ldots, 2n. \end{cases} \tag{2.71}$$

and the covariance weight is:

$$w_i^c = \begin{cases} \frac{\lambda}{n+\lambda} + 1 - \alpha^2 + \beta & \text{if } i = 0, \\ \frac{1}{2(n+\lambda)} & \text{if } i = 1, \ldots, 2n, \end{cases} \tag{2.72}$$

where β is an additional tuning parameter.

Intuitively, this method instantiates one sigma point at the mean and $2n$ points symmetrically spread about the mean. As the state dimension n increases, the sigma points are spread further from the mean and are assigned a lower weight. The α parameter is a scale factor $0 \leq \alpha \leq 1$ that controls the "radius" of the sigma points, with larger values producing a larger spread in the points. The authors of [103] suggest 10^{-3} as a good value for this α. The κ parameter can usually be set to 0, and $\beta = 2$ is the optimal choice for Gaussian problems [101].

Given a set of sigma points $\mathcal{X}$ and weight vectors $W^m = [w_0^m, \ldots, w_{2n}^m]$, $W^c = [w_0^c, \ldots, w_{2n}^c]$, the UKF predict step simply replaces Equations (2.60) and (2.61) with an unscented transform over the (possibly nonlinear) transition function $f(\cdot)$ as:

$$\mathcal{Y} = f(\mathcal{X}), \tag{2.73}$$

$$\bar{x}_{k|k-1} = \sum_{i=0}^{2n} w_i^m \mathcal{Y}_i, \text{ and} \tag{2.74}$$

$$P_{k|k-1} = \sum_{i=0}^{2n} w_i^c (\mathcal{Y}_i - \bar{x}_{k|k-1})(\mathcal{Y}_i - \bar{x}_{k|k-1})^T + Q_{k-1}. \tag{2.75}$$

Similarly, the update step first transforms the predicted sigma points $\mathcal{Y}$ into measurement space using the (possibly nonlinear) measurement function $h(\cdot)$:

$$\mathcal{Z} = h(\mathcal{Y}). \tag{2.76}$$

The mean and covariance of the predicted density can be computed in measure-

ment space using an unscented transform:

$$\bar{z} = \sum_{i=1}^{n} w_i^m \mathcal{Z}_i, \text{ and} \tag{2.77}$$

$$P_z = \sum_{i=1}^{n} w_i^c (\mathcal{Z}_i - \bar{z})(\mathcal{Z}_i - \bar{z})^T. \tag{2.78}$$

The measurement residual follows as:

$$y = z - \bar{z}, \tag{2.79}$$

and the Kalman gain is given by:

$$K = P_{xz} P_z^{-1}, \tag{2.80}$$

where P_{xz} is the cross covariance between the prediction and measurement:

$$P_{xz} = \sum_{i=0}^{2n} w_i^c (\mathcal{Y}_i - \bar{x}_{k|k-1})(\mathcal{Z}_i - \bar{z})^T. \tag{2.81}$$

Finally, the posterior distribution is a multivariate Gaussian with mean and covariance given by:

$$\bar{x}_{k|k} = \bar{x}_{k|k-1} + Ky, \text{ and} \tag{2.82}$$

$$P_{k|k} = P_{k|k-1} - K P_z K^T \tag{2.83}$$

Therefore, the multivariate Gaussian distribution is again a conjugate prior density for the UKF.

The UKF has several desirable properties for practitioners. It is highly robust to nonlinear behavior and discontinuities compared to the EKF [99]. Addition-

ally, it can be applied to arbitrary transition and measurement models with no modification to the algorithm itself. The UKF is also computationally efficient. Although its computational complexity is greater than that of the traditional Kalman filter by a factor of approximately $2n + 1$, it requires significantly less computation than particle filters or other sampling-based methods for nonlinear state estimation [104].

2.4 Multi-Object Tracking

Section 2.3 analyzed the state estimation problem in the single-object case where exactly one object is under track at all times. This section presents a generalized formulation for tracking multiple objects simultaneously. As in the single-object case, the tracker must contend with measurement noise, missed detections, and uncertain data associations due to clutter measurements and false alarms. In the multi-object case, data association becomes considerably more complex since measurements may come from any of the objects instead of only one. This challenge is further compounded by the fact that the number of objects n is often unknown and potentially time-varying, so the tracker must estimate n alongside the individual object states. This section presents a system model based on finite set statistics which provides a natural representation for multi-object tracking dynamics. The set-valued representation is provably Bayes optimal, making it possible to develop statistically grounded models for track initiation, object existence, and clutter processes [16].

Figure 2.8 presents a simple multi-object tracking scenario. In this case, a moving vehicle equipped with a ranging sensor (left) must track the states of other vehicles on the road. The sensor has a limited field-of-view (FOV), so survival and birth models must be used to account for objects entering or exiting

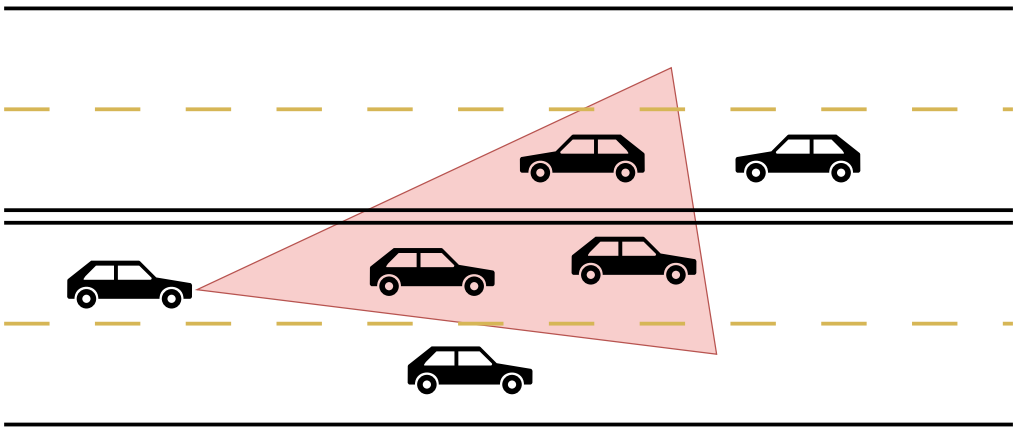


Figure 2.8: Simple multi-object tracking scenario with a vehicle-mounted sensor. The tracker associated to the sensor must content with dynamic object appearance/disappearance, data association ambiguity, missed/false detections, and measurement noise.

the sensor footprint. If sensor measurements do not provide identity information, the tracker must resolve data association ambiguity to identify which cars produced each measurement. The tracker must also contend with missed and false detections (e.g., from non-vehicle objects in the FOV).

2.4.1 Finite Set Statistics

The fields of finite set statistics and point process theory provides a mathematical language for modeling systems whose observations are variable-length finite sets [15]. A random finite set (RFS) $\mathtt{X}$ is a random variable whose realizations are sets of random vectors $\mathcal{X} = \{x^{(1)}, \ldots, x^{(n)}\}$ with $x^{(1)}, \ldots, x^{(n)} \in \mathbb{R}^d$ [16] and whose cardinality $n = |\mathtt{X}|$ is a discrete random variable whose pmf is given by $\rho(n) = P\{|\mathtt{X}| = n\}$. The statistics of an RFS are characterized by a pdf of the form:

$$f(\mathcal{X}) = f(\{x^{(1)}, \ldots, x^{(n)}\}) = n!\rho(n)f_n(x^{(1)}, \ldots, x^{(n)}), \tag{2.84}$$

where $f_n(\cdot)$ is a joint pdf over n random variables that is invariant to the ordering of its inputs. The remainder of this section introduces several families of set-valued distributions that commonly arise in the RFS tracking literature.

The Poisson RFS, also known as a Poisson point process (PPP), is the standard model for clutter and newly-appearing objects in RFS tracking techniques [16]. The cardinality distribution of a Poisson RFS is Poisson distributed with mean μ:

$$\rho_{\text{PPP}}(n) = e^{-\mu}\mu^n/n!, \tag{2.85}$$

and its individual elements are i.i.d. according to a "spatial pdf" $s(x)$, such that a set of size n has the joint pdf below:

$$f_n(x^{(1)}, \ldots, x^{(n)}) = \prod_{x \in \mathcal{X}} s(x). \tag{2.86}$$

The complete pdf for a Poisson RFS follows directly from substituting Equations (2.85) and (2.86) into Equation (2.84):

$$f_{\text{PPP}}(\mathcal{X}) = e^{-\mu} \prod_{x \in \mathcal{X}} \mu s(x). \tag{2.87}$$

It is often more convenient to characterize a Poisson RFS by its intensity function $\lambda(x) = \mu s(x)$. In multi-object tracking application, a Poisson intensity often describes the expected number of objects or false alarms at each point in the state space.

A Bernoulli RFS describes a set that is empty with probability $1 - r$ and contains a single element with probability r. When the set is non-empty, the element has spatial pdf $s(x)$. The Bernoulli RFS consequently has a pdf of the

form:

$$f_{\mathrm{B}}(\mathcal{X}) = \begin{cases} 1-r & \text{if } \mathcal{X} = \emptyset, \\ r \cdot s(x) & \text{if } \mathcal{X} = \{x\}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.88}$$

Multi-object trackers often use Bernoulli RFSs to model individual tracks whose existence is uncertain (e.g., due to data association ambiguity). In such cases, r is the probability that the object truly exists and $s(x)$ is the state density conditioned on the object's existence (e.g., from a Kalman filter).

A multi-Bernoulli (MB) RFS describes the union of n_b independent Bernoulli RFSs $\mathtt{X}^{(j)}$, $j = 1, \ldots, n_b$ which each possess their own state distributions $s^{(j)}(x)$ and existence probabilities $r^{(j)}$. A single realization of an MB RFS $\mathcal{X}^{\mathrm{MB}} = \{x^{(1)}, \ldots, x^{(n)}\}$ with $n \leq n_b$ has n "active" components and $n_b - n$ components which map to the empty set. Define a mapping function $\alpha(\cdot)$ which converts Bernoulli indices $j \in \{1, \ldots, n_b\}$ to "active" indices $\alpha(j) \in \{0, \ldots, n\}$, where each active component maps to a unique value in $\{1, \ldots, n\}$ and all inactive components map to 0. Let $\mathcal{P}^n_{n_b}$ denote the set of all possible index mappings, such that $|\mathcal{P}^n_{n_b}| = n_b!/(n_b - n)!$. The pdf for an MB RFs is the sum of joint pdfs over possible mappings:

$$f(\mathcal{X}) = \sum_{\alpha \in \mathcal{P}^n_{n_b}} \prod_{j=1}^{n_b} f_{\mathrm{B}}^{(j)}(\mathcal{X}^{(\alpha(j))}), \tag{2.89}$$

where $f_B^{(j)}$ is the pdf of the j-th Bernoulli component.

Many multi-object tracking filters maintain explicit track continuity over time by assigning a unique identifier or label $\ell \in \mathcal{L}$ to each track, where $\mathcal{L} = \{\tau(j) : j \in \{1, \ldots, n_b\}\}$ for some labeling function $\tau(\cdot)$. The labeled multi-Bernoulli (LMB) RFS models such cases by augmenting each component in an MB RFS

with an additional label [105]. Realizations of an LMB RFS are sets of *tuples* $\mathcal{X}^{\text{LMB}} = \{(x^{(1)}, l^{(1)}), \ldots, (x^{(n)}, l^{(n)})\}$, where each $x^{(i)}$ is a Bernoulli component. Assuming each label is unique, the mapping between labels and components is fixed and the LMB pdf can be expressed as a product of Bernoulli pdfs:

$$f_{\text{LMB}}(\mathcal{X}) = \prod_{j=1}^{n_b} f_B^{(j)}(\mathcal{X}^{(\tau(j))}). \tag{2.90}$$

The derivations in the remainder of this section also make use of the mathematical tools from multi-object calculus [16]. The set integral of a set-valued function $g(\mathcal{X})$ is defined as:

$$\int g(\mathcal{X})\delta\mathcal{X} = \sum_{n=0}^{\infty} \frac{1}{n!} \int_{\mathbb{R}^d} g(\{x^{(1)}, \ldots, x^{(n)}\}) dx^{(1)} \ldots dx^{(n)}. \tag{2.91}$$

Equation (2.91) integrates over all possible realizations of the input set. The normalization factor $1/n!$ ensures that permutations of a set with the same elements are not integrated multiple times. Assuming each $f_n(\cdot)$ is a valid joint pdf, substituting Equation (2.84) into Equation (2.91) shows that RFS densities must integrate to unity.

2.4.2 Multi-Object Bayesian State Estimation

A Bayesian multi-object tracker applies sequential prediction and update steps to estimate the posterior multi-object density $f(\mathcal{X}_k|\mathcal{Z}_{1:k})$. In this case, $\mathcal{X}_k$ is an RFS representing the multi-object state at time k, $\mathcal{Z}_k$ is an RFS representing the measurement set at time k, and $\mathcal{Z}_{1:k} = \{\mathcal{Z}_1, \ldots, \mathcal{Z}_k\}$ is the sequence of measurement sets up to and including time k. This section presents the Bayesian filtering recursion in its most general form. Evaluating these expressions in closed form is

typically impractical, so multi-object trackers make various approximations and assumptions to make the filtering recursion tractable.

The prediction step propagates the posterior density from time $k-1$ to time k using the RFS Chapman-Kolmogorov Equation:

$$f(\mathcal{X}_k|Z_{1:k-1}) = \int f(\mathcal{X}_k|\mathcal{X}_{k-1}) f(\mathcal{X}_{k-1}|\mathcal{Z}_{1:k-1}) \delta\mathcal{X}_{k-1}, \tag{2.92}$$

where $f(\mathcal{X}_k|\mathcal{X}_{k-1})$ is a multi-object state transition kernel and the integration operation is a set integral. The update step of a Bayesian RFS filter incorporates the new measurement set $\mathcal{Z}_k$ into the predicted density to produce a new posterior via the RFS analog to Bayes' rule:

$$\begin{aligned} f(\mathcal{X}_k|\mathcal{Z}_{1:k}) &= \frac{f(\mathcal{Z}_k|\mathcal{X}_k) f(\mathcal{X}_k|\mathcal{Z}_{1:k-1})}{\int f(\mathcal{Z}_k|\mathcal{X}_k) f(\mathcal{X}_k|\mathcal{Z}_{1:k-1}) \delta X_k} \\ &\propto f(\mathcal{Z}_k|\mathcal{X}_k) f(\mathcal{X}_k|\mathcal{Z}_{1:k-1}), \end{aligned} \tag{2.93}$$

where $f(\mathcal{Z}_k|\mathcal{X}_k)$ is the multi-object likelihood function for the current measurement set. Equations (2.92) and (2.93) are the multi-object analogs to Equations (2.58) and (2.59), respectively. As in the single-object case, the prediction step depends only on the multi-object transition density and the previous posterior distribution. The multi-object update step likewise depends only on the current measurement set and the predicted density.

2.4.3 Track Oriented Multi-Bernoulli/Poisson Filter

This work uses the track-oriented marginal Bernoulli/Poisson (TOMB/P) filter[2] from [19], [20] for state estimation and the graph representation discussed in

[2]A Python implementation of the TOMB/P filter and other multi-object tracking tools can be found at `https://github.com/ShaneFlandermeyer/MOTpy`

Chapters 4. The TOMB/P filter decomposes the complete multi-object state RFS $\mathcal{X}$ into a disjoint union $\mathcal{X} = \mathcal{X}^u \uplus \mathcal{X}^d$. Here, $\uplus$ is the disjoint union operator defined such that $X = X_1 \uplus X_2$ if $X = X_1 \cup X_2$ and $X_1 \cap X_2 = \emptyset$ for sets X_1 and X_2.

The $\mathcal{X}^d$ component models objects that have produced at least one measurement and the $\mathcal{X}^u$ component models *undetected* objects that have not been detected but are hypothesized to exist. As will be shown, this explicit model for undetected objects provides useful information for the search task that can be exploited by an autonomous sensing agent. Since $\mathcal{X}^d$ and $\mathcal{X}^u$ are disjoint, the multi-object state density at time k decomposes into a product of two individual pdfs:

$$f(\mathcal{X}_k|\mathcal{Z}_{1:k}) = f(\mathcal{X}_k^d|\mathcal{Z}_{1:k}) f(\mathcal{X}_k^u|\mathcal{Z}_{1:k}), \tag{2.94}$$

where $\mathcal{X}_k^d = \{(x_k, l_k) \in \mathcal{X}_k | \ell_k = 0\}$ is an LMB RFS with label set $\mathcal{L}_k$. $\mathcal{X}_k^d$ contains $|\mathcal{L}_k| = i_k^d$ Bernoulli components, where i_k^d is the number of surviving objects detected up to and including time k. The $\mathcal{X}_k^u$ component is a Poisson RFS with intensity $\lambda_k^u(x_k)$.

Predict Step

During the prediction step, the TOMB/P filter independently propagates the detected and undetected densities to the current time step. This work uses the standard RFS multi-object dynamics model from [16], which makes the following assumptions:

1. Individual object states evolve independently over time according to a single-object Markov transition density $f(x_k|x_{k-1})$.

2. To model object disappearance, an object with state x_{k-1} "survives" and

continues to exist at time k with probability $p_s(x_{k-1})$ and disappears with probability $1 - p_s(x_{k-1})$.

3. New object appearance is modeled as a Poisson RFS with intensity $\lambda_b(x_k) = \mu_b f_b(x_k)$. In other words, the number of new objects at time k is Poisson-distributed with mean μ_b and new object states are i.i.d. according to the spatial pdf $f_b(x_k)$.

Based on the assumptions above, the state transition density for each detected object is a defined as:

$$f(\mathcal{X}_{k,j}^d|(x_{k-1}^{(j)}, l_{k-1}^{(j)})) = \begin{cases} 1 - p_s(x_{k-1}^{(j)}) & \text{if } \mathcal{X}_{k,j}^d = \emptyset, \\ p_s(x_{k-1}^{(j)}) \cdot f(x_k|x_{k-1}^{(j)}) & \text{if } \mathcal{X}_{k,j}^d = \{(x_k^{(j)}, l_{k-1}^{(j)})\}, \\ 0 & \text{otherwise.} \end{cases} \quad (2.95)$$

Note that surviving detected objects retain their original labels across the predict step. Since state transitions are independent, the complete multi-object transition density for the detected objects is the product of the single-object densities

$$f(\mathcal{X}_k^d|\mathcal{X}_{k-1}^d) = \prod_{j=1}^{i_{k-1}^d} f(\mathcal{X}_{k,j}^d|(x_{k-1}^{(j)}, l_{k-1}^{(j)})). \quad (2.96)$$

Applying the prediction step from Equation (2.92) to an LMB RFS with the transition density in Equation (2.96) produces a new LMB RFS with updated parameters:

$$r_{k|k-1}^{(\ell)} = r_{k-1}^{(\ell)} \int p_s(x_{k-1}) s_{k-1}^{(\ell)}(x_{k-1}) dx_{k-1}, \quad (2.97)$$

$$s_{k|k-1}^{(\ell)}(x_k) = \frac{\int f(x_k|x_{k-1}) p_s(x_{k-1}) s_{k-1}^{(\ell)}(x_{k-1}) dx_{k-1}}{\int p_s(x_{k-1}) s_{k-1}^{(\ell)}(x_{k-1}) dx_{k-1}}, \quad (2.98)$$

where $(r_{k-1}^{(\ell)}, s_{k-1}^{(\ell)})$ are the parameters of the ℓ-th LMB component at the previous time step [19].

Each undetected object modeled by $\mathcal{X}_k^u$ comes from one of two sources: they either existed at time $k-1$ and survived to time k or they are detected for the first time at time k. Previously-existing undetected objects are represented by an LMB RFS, where each Bernoulli component has a non-unique dummy label $l_k = 0$:

$$\mathcal{X}_{k-1}^u = \{(x_{k-1}^{(1)}, 0), \ldots, (x_{k-1}^{(i_{k-1}^u)}, 0)\}. \tag{2.99}$$

New objects are represented with a Poisson birth RFS Γ_k with spatial intensity $\lambda_b(x_k) = \mu_b f_b(x_k)$ (assumption 3). Assuming the states of newborn objects are independent from existing objects, the undetected RFS is a set union of two components:

$$X_k^u = \left(\bigcup_{j=1}^{i_{k-1}^u} X_{k,j}^u \right) \cup \Gamma_k. \tag{2.100}$$

It can be shown [20] that the complete multi-object state transition pdf is:

$$\begin{aligned} f(\mathcal{X}_k^u | \mathcal{X}_{k-1}^u) = e^{-\mu_b} \left(\prod_{i=1}^{i_k^u} \mu_b f_b(x_k^{(i)}) \right) \left(\prod_{j=1}^{i_{k-1}^u} (1 - p_s(x_{k-1}^{(j)})) \right) \\ \times \sum_{\alpha \in \mathcal{M}_{i_{k-1}^u}^{i_k^u}} \prod_{j':\alpha(j')} \frac{p_s(x_{k-1}^{(j')}) f(x_k^{(\alpha(j'))} | x_{k-1}^{(j')})}{\mu_b (1 - p_s(x_{k-1}^{(j')})) f_b(x_k^{(\alpha(j'))})}, \end{aligned} \tag{2.101}$$

where α maps existing objects from the previous time step $j \in \{1, \ldots, i_{k-1}^u\}$ to a unique index $\alpha(j) \in \{1, \ldots, i_k^u\}$ if they survive to time k and 0 if they do not survive, and $\mathcal{M}_{i_{k-1}^u}^{i_k^u}$ is a set of all possible mappings.

After the prediction step, $\mathcal{X}_k^u$ remains a Poisson RFS with an updated inten-

sity function:

$$\lambda^u_{k|k-1}(x_k) = \lambda^b_k(x_k) + \int f(x_k|x_{k-1}) p_s(x_{k-1}) \lambda^u_{k-1}(x_{k-1}) dx_{k-1}. \tag{2.102}$$

When the undetected and birth intensities are Gaussian mixtures with N_u and N_b respective components, Equation (2.102) produces a Gaussian mixture with $N_u + N_b$ components. Consequently, most implementations prune or merge the mixture after each filter step to limit the number of components and reduce computational complexity. Alternatively, [106] proposes a sequential Monte Carlo (SMC) implementation of the TOMB/P filter that approximates each density in the TOMB/P filter with a set of weighted particles. Although the SMC implementation requires significantly more computation, it is well-suited for problems with dynamics that are highly nonlinear and/or non-Gaussian.

Update Step

The update step of a Bayesian RFS filter incorporates the new measurement set $\mathcal{Z}_k$ into the predicted density to produce a new posterior via Bayes' rule. The measurement set is defined as:

$$Z_k = \left\{ z_k^{(1)}, \ldots, z_k^{(m_k)} \right\}, \tag{2.103}$$

where m_k is the number of measurements collected at time k. Describing the measurements as a set makes it possible to model situations with a variable number of measurements at each time step (e.g., due to false alarms or missed detections). This work invokes the standard RFS measurement model [16], which makes the following assumptions:

1. Each object produces at most one measurement.

2. Each measurement corresponds to exactly one object.

3. An object with state x_k produces a measurement with probability $p_d(x_k)$ and no measurement with probability $1 - p_d(x_k)$.

4. If an object with state x_k produces a measurement, the measurement is distributed with likelihood $f(z_k|x_k)$.

5. Measurements which originate from objects are statistically independent of each other and of all clutter measurements.

6. False alarms and clutter measurements are modeled as a Poisson RFS with intensity function $\lambda^c(x_k) = \mu_c f_c(x_k)$.

Given the assumptions above, the RFS likelihood for each object j is:

$$f(\mathcal{Z}_{k,j}|\{x_k^j\}) = \begin{cases} 1 - p_d(x_k^{(j)}) & \text{if } \mathcal{Z}_{k,j} = \emptyset, \\ p_d(x_k^{(j)}) f(z_k^{(j)}|x_k^j) & \text{if } \mathcal{Z}_{k,j} = \{z_k\}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.104}$$

Note that Equation (2.104) has an identical form as the single-object measurement likelihood function from Equation (2.54) since measurements are again Bernoulli RFSs with existence probability p_d. Assuming each of the i_k objects in $\mathcal{X}_k$ can produce at most one measurement, the object-originating measurement set is an MB RFS composed from the union of i_k Bernoulli components.

The multi-object measurement model also treats clutter as a time-varying RFS Λ_k. Assuming the number of clutter measurements is Poisson-distributed with mean μ_c and spatial pdf $f_c(z_k)$, Λ_k is a Poisson RFS with intensity $\lambda_c(z_k) =$

$\mu_c f_c(z_k)$. Unlike the Poisson birth RFS Γ_k, the clutter pdf is defined in the measurement space. The overall measurement RFS Z_k is the union of two RFSs:

$$Z_k = \left(\bigcup_{j=1}^{i_k} Z_{k,j} \right) \cup \Lambda_k. \tag{2.105}$$

All components in Z_k are assumed to be independent given the object state RFS X_k. The derivation in [15] shows that the joint multi-object likelihood function is given as follows:

$$f(\mathcal{Z}_k|\mathcal{X}_k) = e^{-\mu_c} \left(\prod_{m=1}^{m_k} \mu_c f_c(z_k^{(m)}) \right) \sum_{\alpha \in \mathcal{M}_{i_k^u}^{m_k}} \prod_{j=1}^{i_k} g(x_k^{(j)}, \alpha(j); z_k). \tag{2.106}$$

Similar to Equation (2.101), the α function maps each object index $j \in \{1, \ldots, i_k\}$ to its corresponding measurement $\alpha(j) \in \{1, \ldots, m_k\}$ or to zero. The expression for $g(\cdot)$ is given as:

$$g(x_k, \alpha(i); z_k) = \begin{cases} \frac{1}{\mu_c} p_d(x_k^{(i)}) f(z_k^{(m)}|x_k^{(i)}) & \text{if } \alpha(i) = m \in \{1, \ldots, m_k\}, \\ 1 - p_d(x_k^{(i)}) & \text{if } \alpha(i) = 0. \end{cases} \tag{2.107}$$

The update step handles detected and undetected objects separately. For undetected objects, the update step reduces the Poisson intensity proportional to the detection probability:

$$\lambda_{k|k}^u(x_k) = (1 - p_d(x_k)) \lambda_{k|k-1}^u(x_k). \tag{2.108}$$

Equation (2.108) reflects the fact that undetected objects are less likely to exist in regions that were illuminated with high-quality sensor dwells. Note that the expression above depends only on the detection model $p_d(\cdot)$ and not on the

realized measurement set $\mathcal{Z}_k$. For the Gaussian mixture representation used in this work, the TOMB/P filter applies Equation (2.108) independently to each mixture component.

Unlike the predict step, the update step for detected objects must account for uncertain associations between measurements and objects. Assume each measurement $m \in \{1, \ldots, m_k\}$ originates from a single object (previously detected or new) or from clutter, and that each object produces at most one measurement per time step. The TOMB/P filter adds a single Bernoulli component with label $\ell = (k, m) \in \mathcal{L}_k^{\text{new}}$ to X_k^d for each measurement. Each new component represents the hypothesis that measurement m originates from a new object that has not been detected in previous time steps. After the update step, the set of LMB labels is thus

$$\mathcal{L}_k = \mathcal{L}_{k-1} \cup \mathcal{L}_k^{\text{new}}, \tag{2.109}$$

where $|\mathcal{L}_k^{\text{new}}| = m_k$.

A data association (DA) hypothesis c_k specifies a mapping between objects and measurements for a given time step. Let $c_k = [c_k^{(\ell)}]_{\ell \in \mathcal{L}_k} = [a_k^T \ b_k^T]^T$, where the sub-vector $a_k = [a_k^{(\ell)}]_{\ell \in \mathcal{L}_{k-1}}$ maps each LMB component to a measurement (or to zero if the object was not detected):

$$a_k^{(\ell)} = \begin{cases} m \in \{1, \ldots, m_k\} & \text{if } \ell \text{ associated to } m, \\ 0 & \text{if } \ell \text{ is not measured.} \end{cases} \tag{2.110}$$

Conversely, $b_k = [b_k^{(\ell)}]_{\ell \in \mathcal{L}_k^{new}}$ maps each measurement to a detected component or

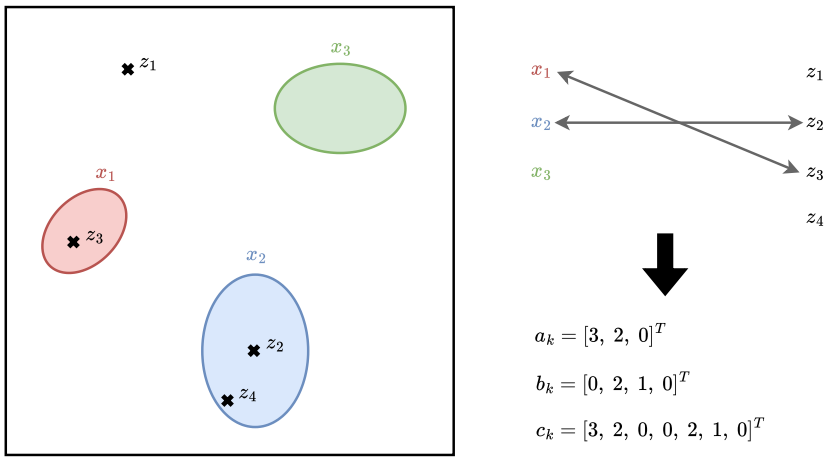


Figure 2.9: Data association representation for an example scenario with three objects and four measurements. (Left) The example scenario. Colored ellipses represent the covariance of the object state estimates and black markers represent measurements. (Right) One possible association hypothesis for the scenario. The DA vector c_k is unique to the given hypothesis.

to zero if the measurement originates from clutter or a new object:

$$b_k^{(\ell=(k,m))} = \begin{cases} \ell' \in \mathcal{L}_{k-1} & \text{if } \ell' \text{ is associated to } m, \\ 0 & \text{if } m \text{ is from a new object or clutter.} \end{cases} \tag{2.111}$$

Figure 2.9 shows a representative scenario along with one feasible data association hypothesis. The scenario contains three existing objects whose covariance ellipsoids are shown along with four measurements. The DA hypothesis assigns measurements z_3 and z_2 to objects x_1 and x_2, respectively, because they fall within the gate region of the state estimates. Though z_4 is also within the gate of x_2, it is not assigned to any object because x_2 has already been associated to z_3. Measurement z_1 lies outside all object gates, so it is also unassigned and is likely to be a clutter measurement or a new object. Object x_3 likewise remains

unassociated because no measurements fall within its gate. The bottom right portion of the figure shows the unique DA vector corresponding to this hypothesis.

The posterior density for detected objects is a weighted sum over DA hypotheses:

$$f(\mathcal{X}_k^d|\mathcal{Z}_{1:k}) = \sum_{c_k} p(c_k) f_{c_k}^{\text{LMB}}(\mathcal{X}_k^d) = \sum_{c_k} p(c_k) \prod_{\ell \in \mathcal{L}_k} f_B^{(\ell, c_k^{(\ell)})}(\mathcal{X}_k^{(\ell)}), \tag{2.112}$$

where $p(c_k)$ is the joint pmf over DA hypotheses that defines the relative weight of DA hypothesis c_k and $f_B^{(\ell, c_k^{(\ell)})}$ is Bernoulli with parameters $(s_k^{(\ell, c_k^{(\ell)})}, r_k^{(\ell, c_k^{(\ell)})})$. The posterior density is a weighted *mixture* of LMB pdfs. Equation (2.112) quickly becomes computationally intractable to evaluate in closed form. The next section presents an approximation that reduces Equation (2.112) to linear time complexity in the number of objects and measurements.

The existence probability and state density can be computed independently for each $f_B^{(\ell, c_k^{(\ell)})}$ in Equation (2.112). Define $D_k^{(\ell)}$ as the probability that object ℓ is detected given its predicted state:

$$D_k^{(\ell)} = \int p_d(x_k) s_{k|k-1}^{(\ell)}(x_k) dx_k, \tag{2.113}$$

and let $C_k^u(z_k^{(m)})$ represent the likelihood that measurement m originates from a previously undetected object:

$$C_k^u = \int f(z_k^{(m)}|x_k) p_d(x_k) \lambda_{k|k-1}^u(x_k) dx_k. \tag{2.114}$$

For previously detected legacy objects, the posterior spatial pdf is given as:

$$s_{k|k}^{(l,a_k^{(\ell)})}(x_k) = \frac{g(x_k, a_k^{(\ell)}; z_k) s_{k|k-1}^{(\ell)}(x_k)}{\int g(x'_k, a_k^{(\ell)}; z_k) s_{k|k-1}^{(\ell)}(x'_k) dx'_k}, \tag{2.115}$$

where $g(\cdot)$ is computed by substituting $a_k^{(\ell)}$ in place of $\alpha(i)$ in Equation (2.107).

Equation (2.115) reduces to RFS Bayes rule for detected objects with $a_k^{(\ell)} \neq 0$. When $a_k^{(\ell)} = 0$, object ℓ does not produce a measurement and the posterior state density is simply the predicted density scaled by $1 - p_d(x_k^{(\ell)})$. The corresponding existence probabilities are:

$$r_k^{(\ell,a_k^{(\ell)})} = \begin{cases} \frac{r_{k|k-1}^{(\ell)}(1-D_k^{(\ell)})}{1-r_{k|k-1}^{(\ell)} D_k^{(\ell)}} & \text{if } a_k^{(\ell)} = 0, \\ 1 & \text{if } a_k^{(\ell)} = \{1, \ldots, m_k\}. \end{cases} \tag{2.116}$$

The first case of Equation (2.116) is a likelihood ratio where the numerator represents the likelihood that the objects exists but was not detected and the denominator represents the hypothesis that the object does not exist or was not detected. The second case is unity because object ℓ produced a measurement and thus exists by definition under hypothesis $a_k^{(\ell)} \neq 0$.

For Bernoulli components representing potential new objects (recall that the update step introduces m_k such components), the existence probabilities are given by:

$$r_k^{(\ell=(m,k),b_k^{(\ell)})} = \begin{cases} \frac{C_k^u(z_k^{(m)})}{C_k^u(z_k^{(m)}) + \lambda_c(z_k^{(m)})} & \text{if } b_k^{(\ell)} = 0, \\ 0 & \text{otherwise,} \end{cases} \tag{2.117}$$

where the first case represents the likelihood that measurement m originates from a new object instead of clutter. The second case is zero because if $b_k^{(\ell)} \neq 0$, measurement m has already been associated to an existing object and thus cannot

correspond to a new object. Since the existence probability for the $b_k^{(\ell)} \neq 0$ case is zero, the spatial pdf is only defined for components with $b_k^{(\ell)} = 0$:

$$s_{k|k}^{(l=(k,m),b_k^{(\ell)}=0)} = \frac{f(z_k^{(m)}|x_k)p_d(x_k)\lambda_{k|k-1}^u(x_k)}{C_k^u(z_k^{(m)})}. \tag{2.118}$$

Equations (2.115)-(2.118) completely characterize the parameters of the posterior state density for all possible DA outcomes. All that remains is to compute the joint DA pmf $p(c_k)$. Since $\mathcal{X}_k^d$ and $\mathcal{X}_k^u$ are disjoint and their individual components are independent, the DA pmf decomposes into a product of belief weights:

$$p(c_k) = p(a_k, b_k) = \left(\prod_{\ell \in \mathcal{L}_{k-1}} \beta_k^{(\ell)}(a_k^{(\ell)}) \right) \left(\prod_{\ell \in \mathcal{L}_k^{\text{new}}} \zeta_k^{(\ell)}(b_k^{(\ell)}) \right). \tag{2.119}$$

Note that Equation (2.119) only holds for "valid" DA configurations in which each object produces at most one measurement and all measurements come from exactly one object or clutter. Configurations where this does not hold are assigned $p(c_k) = 0$. The authors of [19] derive expressions for these association weights, which are given by:

$$\beta_k^{(\ell)}(a_k^{(\ell)}) = \begin{cases} 1 - r_{k|k-1}^{(\ell)} D_k^{(\ell)} & \text{if } a_k^{(\ell)} = 0, \\ r_{k|k-1}^{(\ell)} C_k^{(\ell)}(z_k^{(m)}) & \text{if } a_k^{(\ell)} = m \in \{1, \ldots, m_k\}, \end{cases} \tag{2.120}$$

and

$$\zeta_k^{(\ell=(k,m),b_k^{(\ell)})} = \begin{cases} C_k^u(z_k^{(m)}) + \lambda_c(z_k^{(m)}) & \text{if } b_k^{(\ell)} = 0, \\ 1 & \text{if } b_k^{(\ell)} \neq 0. \end{cases} \tag{2.121}$$

The next section presents an approximate update step that considerably reduces the computational complexity of the TOMB/P update step.

Efficient Computation of the TOMB/P Update Step

The number of feasible DA hypotheses grows exponentially over time. As a result, Equation (2.112) quickly becomes infeasible to evaluate in closed form. This section derives an approximation that reduces the LMB mixture in Equation (2.112) to a single LMB, preventing a combinatorial explosion in the number of components.

Assuming the DA hypotheses for each object are independent, the joint DA pmf can be decomposes into a product of marginal single-object pdfs:

$$p(c_k) \approx \prod_{\ell \in \mathcal{L}_k} p(c_k^{(\ell)}). \tag{2.122}$$

Each single-object pmf is computed by marginalizing the joint pmf:

$$p(c_k^{(\ell)}) = \sum_{\sim c_k^{(\ell)}} p(c_k). \tag{2.123}$$

where $\sim c_k^{(\ell)}$ indicates that the summation is performed over all components of c_k that do not involve $c_k^{(\ell)}$. Substituting this approximation into the expression for the posterior density in Equation (2.112) yields:

$$\begin{aligned} f(\mathcal{X}_k^d | \mathcal{Z}_{1:k}) &= \sum_{c_k} p(c_k) f^{\mathrm{LMB}}(\mathcal{X}_k^d) \\ &= \sum_{c_k} p(c_k) \prod_{\ell \in \mathcal{L}_k} f^{(\ell, c_k^{(\ell)})}(\mathcal{X}_k^{(\ell)}) \\ &= \sum_{c_k} \prod_{\ell \in \mathcal{L}_k} p(c_k^{(\ell)}) f^{(\ell, c_k^{(\ell)})}(\mathcal{X}_k^{(\ell)}) \\ &= \prod_{\ell \in \mathcal{L}_k} \sum_{c_k^{(\ell)}} p(c_k^{(\ell)}) f^{(\ell, c_k^{(\ell)})}(\mathcal{X}_k^{(\ell)}) \\ &= \prod_{\ell \in \mathcal{L}_k} f^{(\ell)}(\mathcal{X}_k^{(\ell)}), \end{aligned} \tag{2.124}$$

where the second equality follows from the definition of an LMB pdf in Equation (2.90), the third follows from Equation (2.122), and

$$f^{(\ell)}(\mathcal{X}_k^{(\ell)}) = \sum_{c_k^{(\ell)}} p(c_k^{(\ell)}) f^{(\ell,c_k^{(\ell)})}(\mathcal{X}_k^{(\ell)}) \tag{2.125}$$

is a normalized sum of labeled Bernoulli densities that is consequently also Bernoulli [16]. Thus, Equation (2.124) is an LMB density and the posterior reduces to an LMB-Poisson RFS.

The parameters of the Bernoulli pdf $f^{(\ell)}(\mathcal{X}_k^{(\ell)})$ are computed separately for legacy and new components. Objects detected prior to time k have the following parameters:

$$r_k^{(\ell)} = \sum_{a_k^{(\ell)}=0}^{m_k} p(a_k^{(\ell)}) r_k^{(\ell,a_k^{(\ell)})}, \text{ and} \tag{2.126}$$

$$s_k^{(\ell)}(x_k) = \frac{1}{r_k^{(\ell)}} \sum_{a_k^{(\ell)}=0}^{m_k} p(a_k^{(\ell)}) r_k^{(\ell,a_k^{(\ell)})} s_k^{(\ell,a_k^{(\ell)})}(x_k). \tag{2.127}$$

New Bernoulli components introduced at time k only have non-zero existence probability when their corresponding measurement is not associated to an existing object. In this case, the Bernoulli parameters are given by:

$$r_k^{(\ell)} = p(b_k^{(\ell)} = 0) r_k^{(\ell,b_k^{(\ell)}=0)}, \text{ and} \tag{2.128}$$

$$s_k^{(\ell)}(x_k) = s_k^{(\ell,b_k^{(\ell)}=0)}(x_k). \tag{2.129}$$

These expressions require DA pmfs $p(a_k)$ and $p(b_k)$, which can be computed in linear time using the sum-product algorithm (SPA) over a factor graph. Though the SPA-based data association method is used in the TOMB/P filter implementation developed for this work, its derivation is omitted here and the interested

reader is referred to [19], [20] for details.

2.5 Radar Signal Model

This section outlines the pulsed radar signal model assumed for the cognitive spectrum sharing agent presented in Chapter 3. Consider a pulse-agile radar system that transmits and receives a continuous train of pulses, with each pulse temporally separated by a constant pulse repetition interval (PRI). The time delay between the transmission of a pulse and its received echo provides ranging information about objects in the scene. The successive phase changes between pulses due to object motion provides an estimate of the Doppler shift and therefore radial velocity of the objects.

When transmitting a pulse, the radar generates a potentially-unique waveform $x_i(t)$ and mixes it to a carrier frequency f_c which is suitable for the application space at hand, yielding the time-domain transmitted signal:

$$x_{RF}(t) = x_i(t)e^{j2\pi f_c t}. \tag{2.130}$$

One of the most common choices for the transmit signal is the linear frequency modulated (LFM) waveform of the form below:

$$x_{\text{LFM}}(t) = a(t)\exp\left[j2\pi\left(-\frac{B}{2}t + \frac{B}{2T_p}t^2\right)\right], \quad t \in [0, T_p], \tag{2.131}$$

where $a(t)$ is the amplitude, B is the signal bandwidth and T_p is the pulse duration [107]. Figure 2.10 shows the time domain waveform of an LFM with $B = 10\,\text{MHz}$ and $T_p = 10\,\mu\text{s}$, oversampled by a factor of 4. Note that the amplitude of the waveform is constant throughout (neglecting windowing or hard-

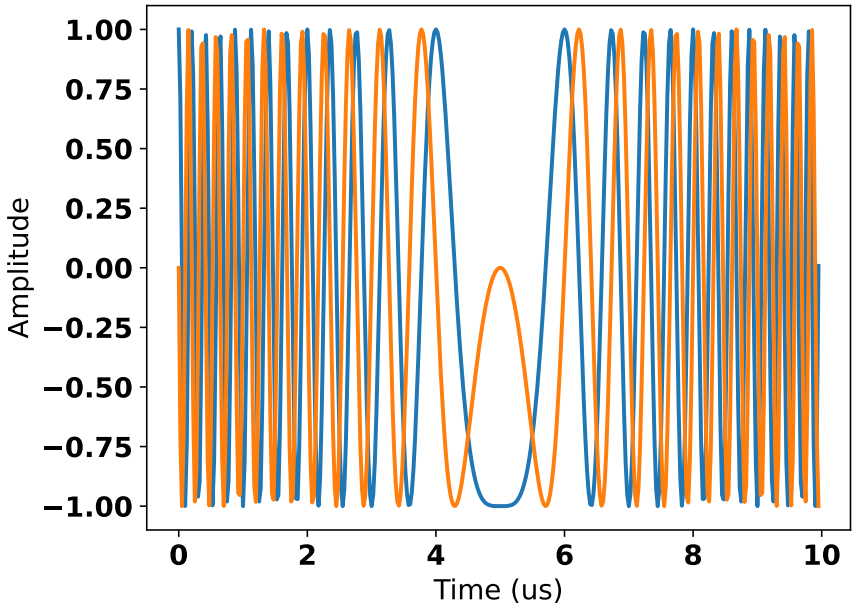


Figure 2.10: LFM waveform in the time domain for $B = 10\,\text{MHz}$ and $T_p = 10\,\mu\text{s}$. The blue curve represents the real component of the signal and the orange curve represents the imaginary component.

ware distortion). Figure 2.11 shows the frequency spectrum of a 10 MHz LFM waveform for various values of the time-bandwidth product BT_p. The LFM's frequency spectrum becomes increasingly rectangular as the time-bandwidth product increases. Therefore, spectrum sharing applications commonly assume the radar emission is perfectly rectangular as a first-order approximation.

For any choice of waveform $x_i(t)$, the signal received from a scatterer in the environment is a delayed, scaled, and Doppler-shifted copy of the transmitted signal:

$$y_{RF}(t, m) = \alpha(m) x_i(t - \tau(m)) e^{j2\pi(f_c + F_D)(t - \tau(m))}, \tag{2.132}$$

where $\alpha(m)$ is the amplitude scale factor computed from the radar range equation, $\tau(m)$ is the two-way propagation delay between the object and radar for each pulse, and F_D is the Doppler shift imparted from to the relative radial mo-

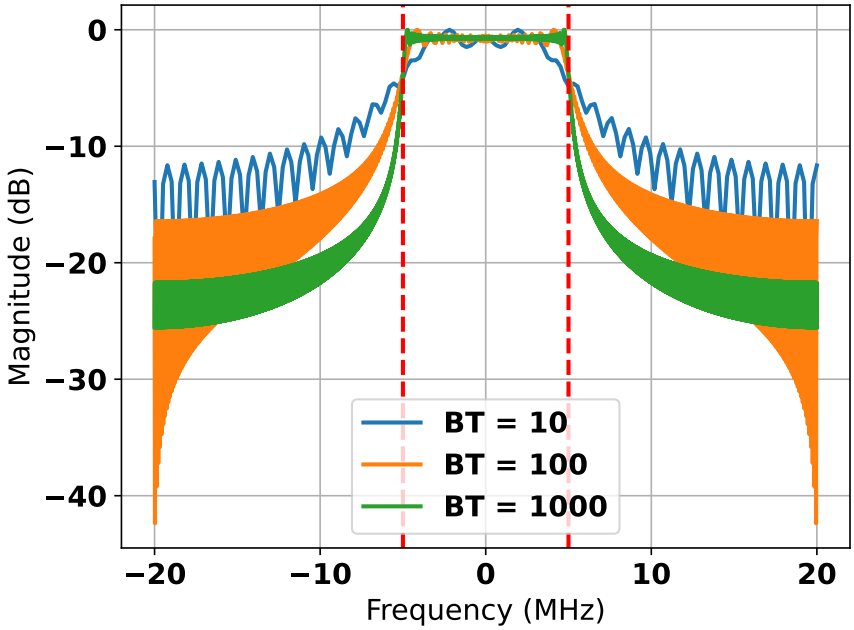


Figure 2.11: Frequency spectrum of a 10 MHz LFM waveform for various values of the time-bandwidth product. The spectral mask becomes asymptotically rectangular as the time-bandwidth product increases.

tion between the object and radar [107]. Before processing the received data, the radar down-converts the received signal from the RF carrier to get the baseband signal:

$$y_{BB}(t,m) = \alpha(m)x_i(t-\tau(m))e^{j2\pi[F_D(t-\tau(m))-f_c\tau(m)]}. \tag{2.133}$$

Next, range information is extracted by performing pulse compression on the received signal using a matched or mismatched filter $h(t)$. The pulse compression output is obtained by convolving the baseband received signal with $h(t)$:

$$y_{MF}(t,m) = y_{BB}(t,m) * h(t). \tag{2.134}$$

The traditional matched filter is designed to maximize the signal-to-noise ratio (SNR) at the filter output. The matched filter impulse response is the time-

reversed, complex conjugate of the transmitted waveform:

$$h_i(t) = \beta x_i^*(T_m - t), \tag{2.135}$$

where β is a scalar and T_m is a delay introduced to make the filter causal [107]. This work sets $\beta = 1$ and T_m to the duration of the waveform.

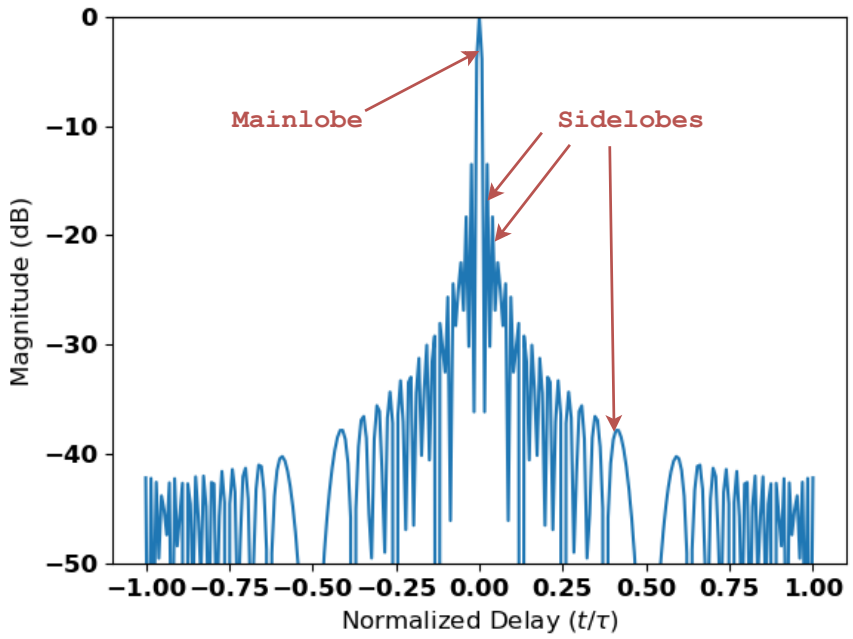


Figure 2.12: Pulse compression response for an LFM waveform. The matched filter produces a narrow peak at the received signal delay (the mainlobe) whose width is based on the range resolution of the radar. It also produces sidelobes that may mask the presence of other objects in the scene.

Figure 2.12 shows an example of the pulse compression output for an object with zero delay. The radar transmits an LFM waveform with a time-bandwidth product $BT_p = 64$, producing a narrow peak at zero delay known as the mainlobe. The width of the mainlobe defines the range resolution, which is inversely proportional to the waveform bandwidth and determines how closely two objects can be spaced in range and still be resolved. As a result of the pulse compression

process, the matched filter response also produces undesirable range sidelobes that we wish to minimize during receive processing.

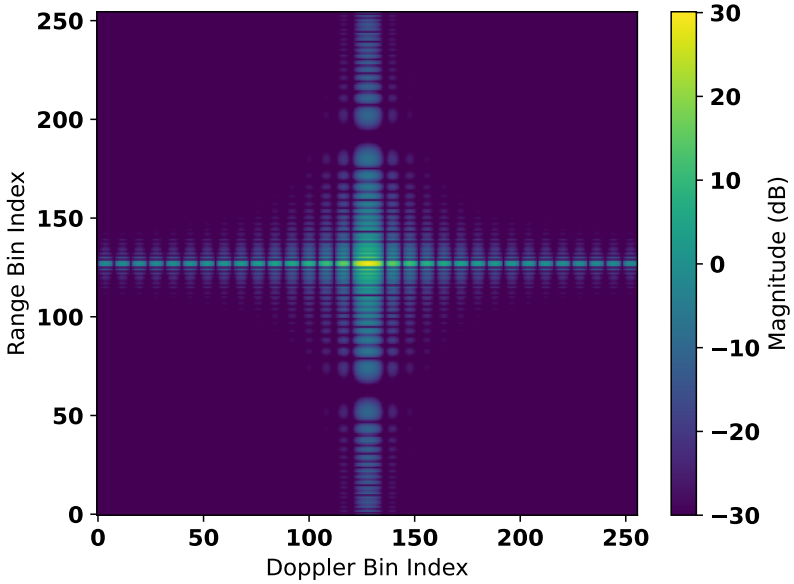


Figure 2.13: Range-Doppler response of an LFM after pulse-Doppler processing (pulse compression and a slow-time DFT) using $N_{CPI} = 32$ pulses. Applying pulse-Doppler processing to a coherent train of pulses produces a PSF with a peak at the range and Doppler of the object, but includes sidelobes in both dimensions.

Following pulse compression, the pulse-Doppler processor creates a coherent processing interval (CPI) from N_{cpi} consecutive pulses and coherently integrates them using a discrete fourier transform (DFT) to form a range-Doppler map (RDM). Figure 2.13 shows the range-Doppler point spread function (PSF) for an LFM waveform after applying Doppler processing on a 32-pulse CPI. In pulse-Doppler processing, the range/delay axis is often referred to as fast-time and the pulse axis is referred to as slow-time. The range sidelobes in Figure 2.12 manifest as a vertical stripe in the PSF, and sinc-like sidelobes are introduced in Doppler due to Doppler processing. This ideal PSF is only obtained if the pulses being

integrated use the same transmit waveform $x_i(t)$ to maintain full coherence. A deleterious effect known as range sidelobe modulation (RSM) occurs when non-identical pulses are processed in a CPI [108].

Chapter 3

Cognitive Radar Spectrum Sharing

This chapter analyzes the radio frequency (RF) spectrum sharing problem from a cognitive radar perspective. The scenario of interest is illustrated in Figure 3.1. Here, a pulse-agile cognitive radar system is deployed in a congested spectrum environment. The radar is a secondary user that must optimize its emissions on each pulse to avoid interference with primary users in the frequency band of interest. At the same time, the radar aims to maximize its waveform bandwidth and minimize pulse-to-pulse variations in its waveform to produce high-quality data products.

Figure 3.2 shows a representative example of the spectrum observations produced by a radar operating in the 2.4-2.5 GHz band. The data was collected from USRP X310 software-defined radio (SDR) operating using 100 MHz front-end bandwidth at a center frequency of 2.45 GHz. In this case, the radar transmits an LFM waveform whose spectrum is approximately rectangular (yellow lines) at some pulse repetition interval (PRI). Ideally, the radar should occupy as much bandwidth as possible without colliding with the other users in the channel (red bounding boxes). To this end, this chapter formulates the cognitive radar spectrum sharing problem as a partially-observable Markov decision process (POMDP) and presents a neural network architecture for processing

spectrum observations.

3.1 Proposed Algorithm

This section presents the reinforcement learning formulation for the proposed cognitive agent. For each transmitted pulse, the agent adapts the radar waveform parameters to jointly optimize the following competing objectives:

- Objective 1: Maximize the bandwidth of each transmitted pulse to achieve sufficient range resolution.
- Objective 2: Minimize mutual interference between the radar and other users in the spectrum to improve SINR and meet spectrum allocation requirements.
- Objective 3: Minimize intra-CPI changes to the transmitted waveform to mitigate the severity of distortions such as range sidelobe modulation (RSM), which arise when standard pulse-Doppler processing is applied to non-identical pulses [108].

The algorithm developed in this section was published in IEEE Transactions on Radar Systems (T-RS) [109]. Much of the results and analysis in this chapter follow from the journal publication.

3.1.1 State Representation

The observations presented to the RL agent are generated as follows: the fast spectrum sensing (FSS) energy detector from [23] is applied to produce a binary channel representation $x_t \in \{0,1\}^{N_{FFT}}$ for the N_{snap} most recent snapshots,

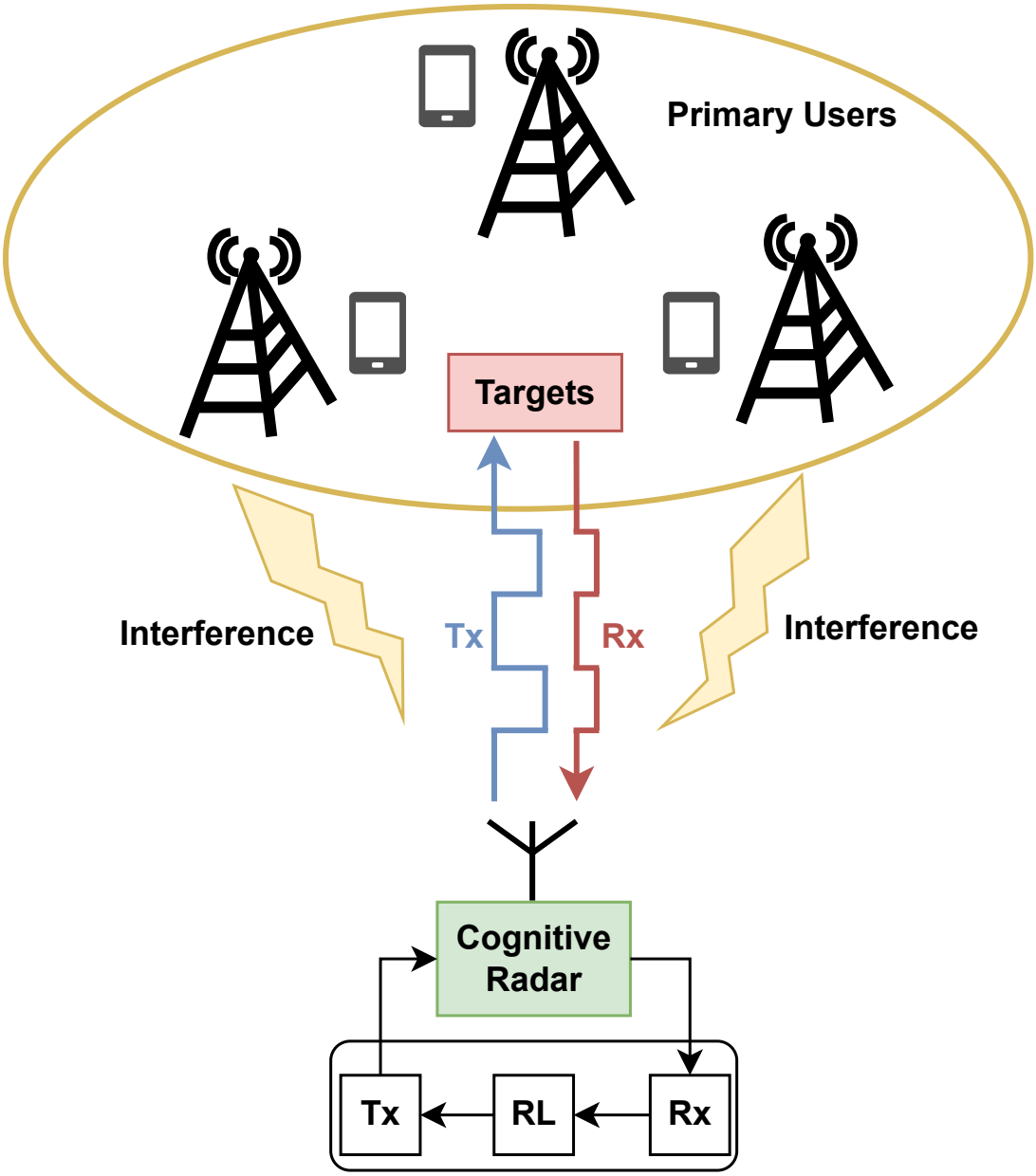


Figure 3.1: Cognitive radar operational environment. The pulsed radar system is a secondary user in a congested spectrum environment, so it must adapt its emissions to reduce mutual interference while maximizing detection and tracking performance.

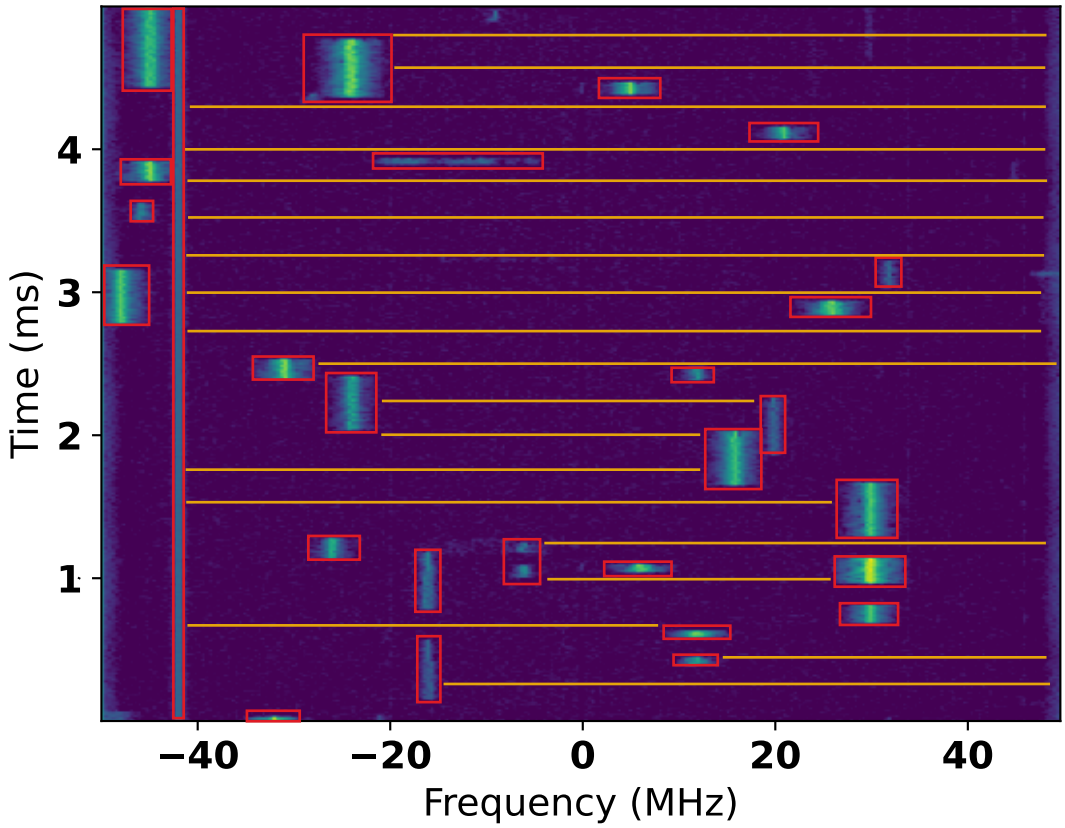


Figure 3.2: Example spectrogram in the 2.4-2.5 GHz band. Red bounding boxes indicate portions of the frequency spectrum occupied by other users. Yellow lines indicate the widest open band available for occupation by a radar system.

where the FSS detection threshold is adaptively computed using the hardware-optimized cell averaging estimation (HO-CAE) algorithm proposed in [110]. Unlike previous approaches which heavily down-sample the input spectrum to 5-10 bins [37], [39] for computational tractability, the observations presented to the agent in this work maintain full resolution in time and frequency as an $N_{snap} \times N_{FFT}$ matrix.

3.1.2 State Encoder Network

Figure 3.3 illustrates the neural network architecture developed for the cognitive spectrum sharing agent. The network uses a recurrent-attention structure that is tailored for pulsed radar operation and operates as follows: first, the agent collects the previous N_{snap} observations as described in Section 3.1.1, then independently projects each observation into a learned embedding space of size N_{embed}. Following a sinusoidal positional encoding stage, the embedding vectors are passed into a multi-head attention (MHA) layer to extract relational information on short intra-pulse time scales. Information is further compressed by a pooling stage (e.g., max pooling), then passed to an LSTM and final fully-connected layer.

Unlike the MHA block, the LSTM encodes long-term relationships that span across many pulses. In addition to the attention output, the LSTM also receives the agent's previous action a_{t-1} and the index i of the next pulse within the current CPI (normalized to the range $[0, 1]$). Finally, the LSTM output is passed through a single fully-connected dense layer to produce the final action a_t and value V_t for the RL agent. To improve training stability, the actor and critic networks are implemented as separate copies of this architecture that do not share parameters or layers. This approach therefore trains two separate networks, but

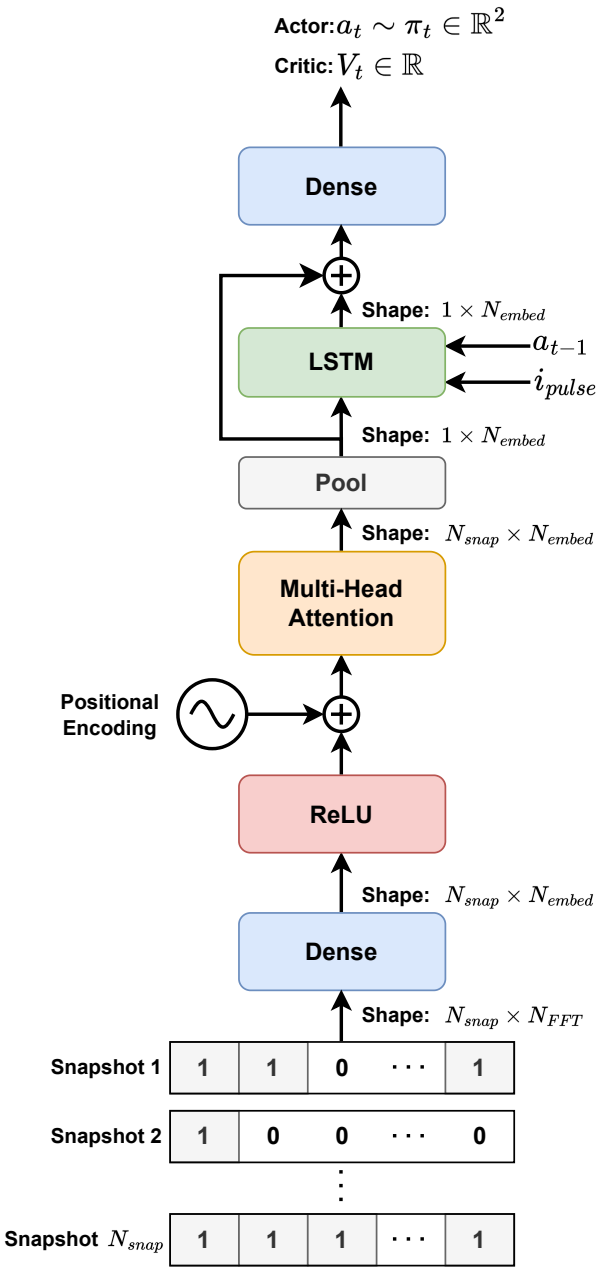


Figure 3.3: Recurrent-attention neural network architecture for the cognitive agent. The full-resolution observations following each pulse are projected into a learned embedding space, then the multi-head attention module identifies patterns at intra-pulse timescales. This information is further compressed via a pooling operation, then passed to an LSTM module which learns high-level patterns across pulses.

only the actor network is required for inference after training is complete.

3.1.3 Action Space

At the start of each PRI, the agent adapts the frequency-domain characteristics of its transmitted waveform to jointly optimize the objectives outlined at the beginning of this section. The radar transmits a waveform whose spectrum is approximately rectangular and contiguous (e.g., a linear FM waveform). Under this assumption, the agent's actions are parameterized by a vector $a_t \in \mathbb{R}^2$ which specifies the start and stop frequency of the waveform's spectral mask. Unlike existing approaches that discretize the action space into discrete bins, this agent operates with a action space that can produce waveforms with arbitrarily precise bandwidths and center frequencies. Consequently, this action representation scales to arbitrarily wideband channels since it is agnostic to the channel bandwidth and FFT size. Each action is normalized to the range $[-1, 1]$, with -1 corresponding to the lowest valid frequency in the channel and 1 to the highest such that a waveform defined by $a_t = [a_{start}, a_{stop}] = [-1, 1]$ occupies the entire channel bandwidth. The agent samples a_{start} and a_{stop} from independent components of a multivariate Gaussian distribution whose output is scaled to the appropriate range using the hyperbolic tangent function as described in [111]. If the agent selects $a_{start} > a_{stop}$, the radar transmits a waveform with a user-specified minimum bandwidth whose spectral mask begins at a_{start}.

3.1.4 Reward Function

This section derives a reward function that encourages the agent to achieve the objectives outlined at the beginning of this section. The reward function has two

components: the first encourages the agent maximize its bandwidth utilization while minimizing collisions with other users:

$$R_{spectrum} = (B_r - B_{widest}) - \alpha_c B_c, \tag{3.1}$$

where B_r is the radar waveform bandwidth, B_{widest} is the bandwidth of the widest unoccupied band at time t, and B_c is the collision bandwidth that is simultaneously occupied by the radar and another user. Each of these quantities is expressed in FFT bins, although they are normalized to the range $[0, 1]$ in the final reward computation by dividing by the total number of FFT bins. The B_{widest} term is extracted from the widest open band identified by the HOCAE-FSS algorithm, and the collision bandwidth B_c is computed as:

$$B_c = \sum_{i=1}^{N_{FFT}} \mathbb{1}\left\{x_r[i] = x_{int}[i]\right\}, \tag{3.2}$$

where $\mathbb{1}\{\cdot\}$ is an indicator function that is unity when the condition in brackets is met and zero otherwise. The vectors x_r and x_{int} are length-N_{FFT} binary detection vectors for the radar and interference, respectively. These vectors equal 1 in occupied bins and 0 in unoccupied bins.

The α_c parameter determines the agent's preference for collision avoidance over bandwidth utilization. A small α_c value encourages the agent to aggressively pursue open spectrum, while a large α_c causes the agent to act more conservatively in order to avoid mutual interference. The α_c parameter can also be interpreted as the number of bins of radar bandwidth that the agent is willing to sacrifice to avoid a collision in a single bin. Though this chapter trains separate agents for various values of α_c, practical implementations could train one agent over a range of α_c values and adjust the parameter in real time to meet

operational requirements.

The second component of the reward function mitigates distortions in the range-Doppler response caused by adapting the transmitted waveform on each pulse. As discussed in Chapter 2.5, pulse-Doppler radar processing assumes all pulses are identical within a processing interval and applies a DFT across pulses to extract Doppler information. Violating this assumption causes range sidelobe modulation (RSM) effects [112] that significantly reduce detection sensitivity. Though algorithms for processing non-identical pulses exist [108], [113]–[115], they are generally computationally intractable and difficult to incorporate into real-time processing. It is more desirable to eliminate these distortions in the data itself by minimizing pulse-to-pulse waveform adaptation during each coherent processing interval (CPI) [116]. Consequently, the second reward term *penalizes* the agent for modifying its waveform parameters within a CPI:

$$R_{adapt} = -\left(\beta_{bw}|B_r - \mu_{B_r}| + \beta_{f_c}|f_c - \mu_{f_c}|\right), \tag{3.3}$$

where $|\cdot|$ is the absolute value operator, μ_{B_r} and μ_{f_c} are the mean bandwidth and center frequency for all waveforms in the current CPI, and β_{bw} and β_{f_c} are scale factors that determine the relative importance of each component. Similar to Equation (3.1), each component in Equation (3.3) is expressed in FFT bins and normalized to the range $[0, 1]$ in the final reward computation. Equation (3.3) is inspired by the distortion function introduced in [39], but Equation (3.3) considers context from the entire CPI rather than only the previous pulse.

The complete reward function is a normalized sum of Equations (3.1) and (3.3):

$$R = \frac{1}{N_{FFT}}\left(R_{spectrum} + R_{adapt}\right). \tag{3.4}$$

The values of α_c, β_{bw}, and β_{f_c} can be tuned to produce various agent behaviors that prioritize different objectives. Though each parameter has a straightforward meaning, care must be taken to ensure that the penalty values do not dominate the reward function and prevent the agent from learning useful radar behavior. The experiments sweep across several combinations of these parameters to examine their impact on radar and spectrum sharing performance.

3.2 Experiment Setup

3.2.1 Hyperparameters and Baseline Comparisons

Section 3.3 evaluates the RL spectrum sharing agent using over-the-air data collected from the 2.4-2.5 GHz and 2.59-2.69 GHz bands in an indoor campus environment. All spectrum data is collected from a USRP X310 SDR platform using 100 MHz of instantaneous bandwidth in a receive-only mode. Tables 3.1 and 3.2 summarize the experimental parameters used for the RL agent and radar system, respectively. When values vary across experiments, the specific value is indicated in the text.

The experiments that follow compare the behavior of the RL agent from Section 3.1 with two other agents in the spectrum environments of interest. The first baseline comparison is a sense-and-avoid (SAA) agent that selects the parameters of an LFM waveform at each time t using the decision logic below:

1. Sense the spectrum using the HOCAE-FSS algorithm to create a length-N_{FFT} binary detection vector for time $t-1$.

2. Locate the position of the widest unoccupied bandwidth at time $t-1$.

3. Synthesize and transmit a waveform whose spectrum fills the open band

Table 3.1: PPO agent parameters

Parameter	Value
Time horizon (T_h)	1024 steps
Num. parallel actors (N_a)	16
Recurrent sequence length	4
Discount factor (γ)	0.8
GAE parameter (λ)	0.95
Policy clip fraction (ϵ)	0.2
Num. gradient epochs	10
Learning rate	2.5×10^{-4}
Collision weight (α_c)	$0 - 30$
Bandwidth distortion factor (β_{bw})	$0 - 1$
center distortion factor (β_{f_c})	$0 - 1$

Table 3.2: Radar system parameters

Parameter	Value
Channel Bandwidth	100 MHz
FFT Size	1024 samples
PRI	204.8 μs
CPI Length	256 pulses
HO-CAE window size (n)	64
HO-CAE order selection (k)	5
P_{FA}	1×10^{-6}

identified in step 2. If the widest open band is smaller than a user-specified minimum bandwidth, the radar transmits a waveform with the minimum bandwidth that occupies the entirety of the widest band.

The SAA agent assumes that the interference environment does not change significantly between individual snapshots and does not attempt to predict future occupancy [41]. This assumption generally holds for users in the 2.4-2.5 GHz and 2.59-2.69 GHz bands which primarily consist of communications systems. As a result, the SAA agent is highly performant in terms of collision avoidance and bandwidth utilization in these environments. The SAA agent also operates with

low latency since HOCAE-FSS is highly optimized for real-time spectrum sensing on FPGAs [110]. However, the SAA agent is unable to exploit temporal patterns in the spectrum environment to further improve performance.

The double deep Q-network (DDQN) architecture derived in [39] serves as a representative baseline for the state-of-the-art RL approaches prior to this work. Unlike the agent derived in this Chapter, the DDQN agent discretizes its spectrum observations and action space into a small number of bins. Though the DDQN agent also uses an LSTM to capture patterns across time steps, it processes only the snapshot immediately before transmission rather than the entire history since the previous pulse. To facilitate a more fair comparison, the experiments that follow discretize the DDQN agent's action space into 10 bins rather than the 5 in the original paper. Both agents use the full frequency resolution (1024 FFT bins) for their input observations and both agents are trained on the joint reward function defined in Equation (3.4).

3.2.2 Evaluation Metrics

This section introduces a set of metrics that characterize the agent's ability to achieve Objectives 1-3 (Section 3.1). Each metric is computed as the average value over episodes of N pulses each. Each expectation in the equations below is computed over 16 independent parallel environments for each agent.

The mean waveform bandwidth over each episode quantifies the agent's performance on the bandwidth-maximization task (Objective 1):

$$\bar{B}_r = \mathbb{E}\left[\frac{1}{N}\sum_{t=0}^{N} B_r(t)\right], \tag{3.5}$$

where $B_r(t)$ is the radar bandwidth from the pulse at time t. Two metrics assess

the agent's interference avoidance capabilities: the missed opportunity bandwidth B_{miss} and collision bandwidth B_c. A missed opportunity occurs when a frequency bin is unoccupied but not used by the radar. Since the radar waveform's frequency spectrum is constrained to be contiguous and rectangular, the missed opportunity bandwidth of each pulse is the difference between the radar bandwidth and B_{widest}, the largest contiguous sub-band that can be occupied without collision:

$$\bar{B}_{miss} = \mathbb{E}\left[\frac{1}{N}\sum_{t=0}^{N}(B_{widest}(t) - B_r(t))\right]. \tag{3.6}$$

It is important to note that (3.6) can be negative when $B_r > B_{widest}$, which is only possible if the radar collides with another user. Consequently, the mean collision bandwidth measures the average fraction of the channel occupied by both the radar and a primary user, or:

$$\bar{B}_c = \mathbb{E}\left[\frac{1}{N}\sum_{t=0}^{N}B_c(t)\right], \tag{3.7}$$

where $B_c(t)$ is the single-step collision bandwidth at time t computed from (3.2).

The agent must also minimize distortions in the range-Doppler response due to pulse-agile operation (Objective 3). The mean change in bandwidth/center frequency over the course of a CPI serves as a proxy for distortion in the range-Doppler domain. The mean change in bandwidth is computed as:

$$\overline{\Delta B} = \mathbb{E}\left[\frac{1}{N}\sum_{t=0}^{N}|B_r(t) - \mu_{B_r}|\right] \tag{3.8}$$

and the mean change in center frequency is:

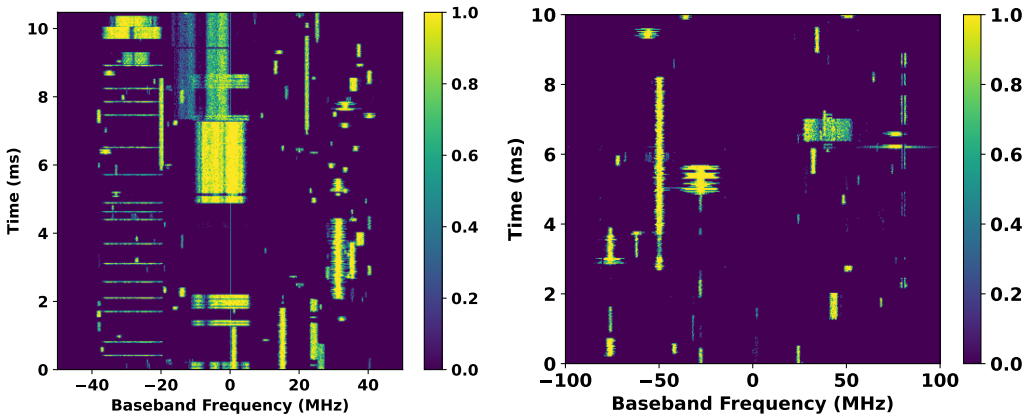
$$\overline{\Delta f_c} = \mathbb{E}\left[\frac{1}{N}\sum_{t=0}^{N}|f_c(t) - \mu_{f_c}|\right], \tag{3.9}$$

where μ_{B_r} and μ_{f_c} are the mean radar bandwidth and center frequency over the entire CPI, respectively. Finally, the analysis that follows reports the cumulative episodic reward to quantify the agent's overall performance on the joint optimization problem outlined in Objectives 1-3.

3.3 Results

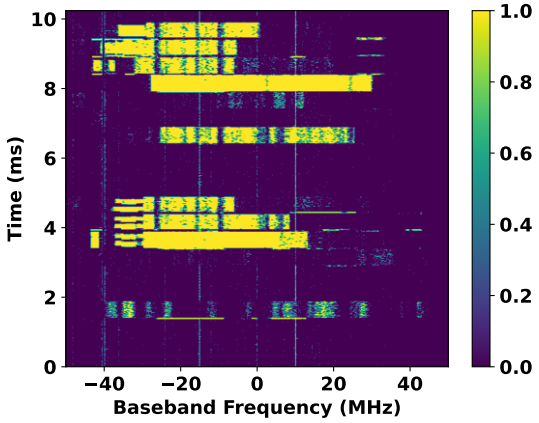
The experiments in this section characterize the RL agent's behavior across a variety of operating conditions using over-the-air data recorded from a USRP X310 SDR platform (Figure 3.4). In the first set of experiments, the agents operate in the 2.4-2.5 GHz industrial, scientific, and medical (ISM) band in an indoor campus environment. As Figures 3.4a and 3.4b show, the spectral characteristics of the emitters in this band vary drastically as a function of time, frequency, and network load, making it a good candidate for evaluating agent performance in realistic scenarios. Similar experiments are conducted in the 2.59-2.69 GHz band, whose sole primary user is a local cellular tower. Figure 3.4c shows that this band is more structured than 2.4-2.5 GHz but exhibits large variability in occupation over time. Thus, these bands serve as useful benchmarks for evaluating agent performance in realistic operational environments.

All recorded data used in the experiments that follow remove FSS detections at -42 MHz and 0 MHz are removed from each snapshot, since those correspond to constant spurious signals from the X310 hardware. The recorded datasets are large enough that experiences are never repeated during training, and separate



(a) 2.4-2.5 GHz campus environment.

(b) 2.4-2.5 GHz low-traffic period.



(c) 2.59-2.69 GHz near a local cellular tower.

Figure 3.4: HO-CAE/FSS detections in scenarios encountered during training.

datasets are used for training and evaluation. Specifically, the measured training dataset consists of approximately 8000 unique CPIs and the evaluation dataset consists of 2000 CPIs. Each experiment is repeated over 5 random seeds.

3.3.1 2.4-2.5 GHz Environment

The first experiment compares the spectrum sharing behavior of the SAA, DDQN, and PPO agents in the 2.4-2.5 GHz ISM band. Since this experiment focuses on the spectrum sharing task, the waveform adaptation penalty from Equation (3.3) is disabled by setting $\beta_{bw} = \beta_{f_c} = 0$. Figures 3.5-3.7b show the performance metrics defined in Section 3.2.2 over the training interval for each agent class. For each RL agent, the solid lines in each plot indicate the mean value of the metric and the shaded regions indicate the upper and lower bounds across all trials. For the SAA agent, dashed horizontal lines indicate the average value of the metric across the entire dataset since SAA is a fixed policy that is not trained.

Figure 3.5 shows the episodic spectrum reward over training in the 2.4-2.5 GHz band. Each agent is trained over 5 random seeds, and the shaded region in the figure indicates the 95% confidence interval.Although both RL agents successfully converge after the training period, the proposed PPO agent obtains a larger episodic reward than the DDQN baseline for the entire training period. The difference between the two approaches is especially pronounced in the early stages of training. Specifically, the PPO agent smoothly converges to its final reward in approximately 500k steps, while the DDQN agent requires more than 1M. The PPO agent also exhibits lower variation in rewards across random seeds compared to the DDQN agent.

To better characterize the performance gap between the agents, Figure 3.6 plots the relevant spectrum sharing metrics over the 2M step training period. The

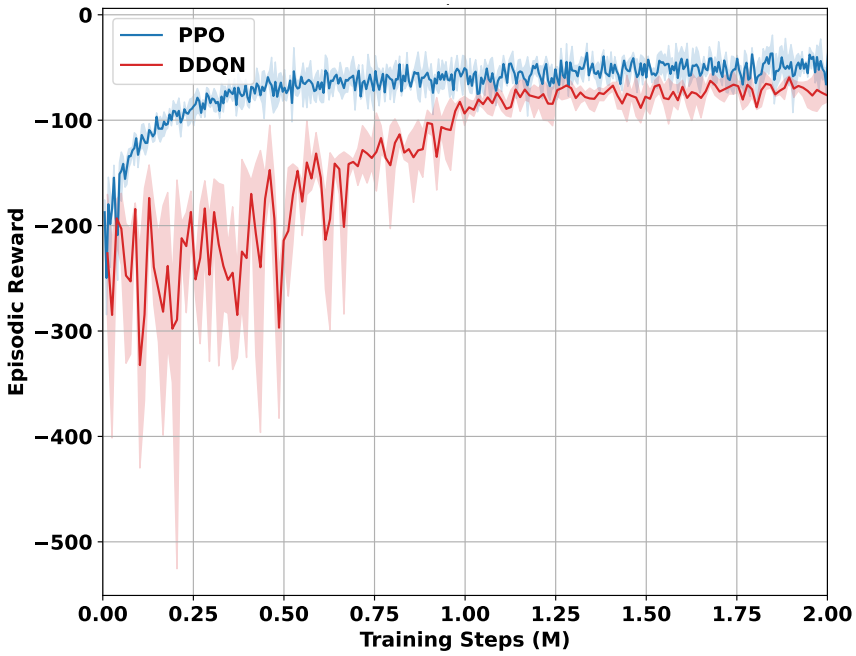
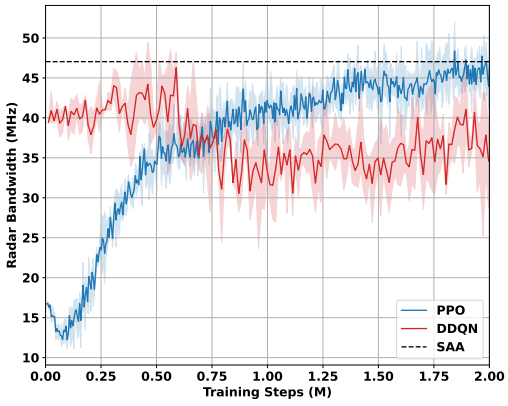


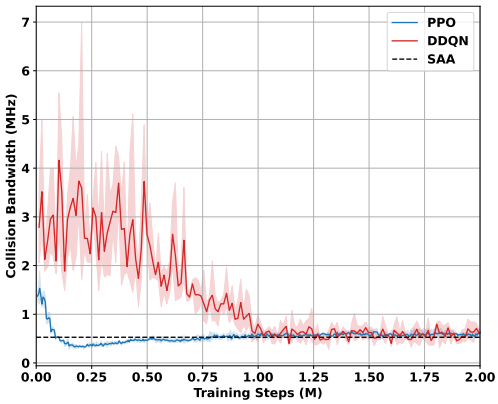
Figure 3.5: Episodic training reward in the 2.4-2.5 GHz ISM band (no waveform adaptation penalty).

DDQN-based cognitive radar initially utilizes large portions of the spectrum with high collision bandwidth, causing significant mutual interference with other users (Figures 3.6a and 3.6b). In contrast, the PPO agent initially uses a smaller radar bandwidth and gradually increases its utilization while maintaining consistently low collision bandwidths. After training, the PPO agent converges to similar spectrum utilization characteristics as the SAA agent. This behavior is highly desirable since the SAA agent is highly performant when maximizing bandwidth and avoiding interference are the primary objectives of interest.

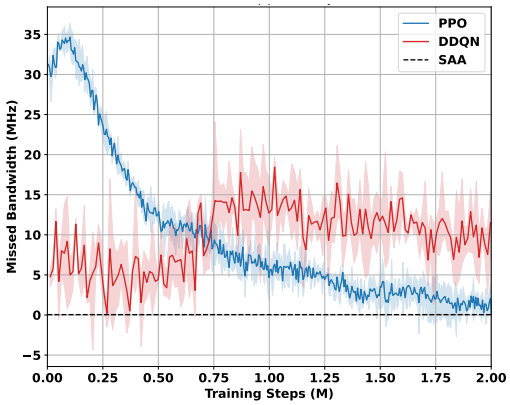
Although the DDQN agent ultimately avoids collisions at a similar rate to the other two agents, its bandwidth utilization actually *decreases* over training. Ultimately, the DDQN agent utilizes approximately 10 MHz less bandwidth per pulse than the PPO or SAA agents, often missing large open bands in the spec-



(a) Mean radar waveform bandwidth

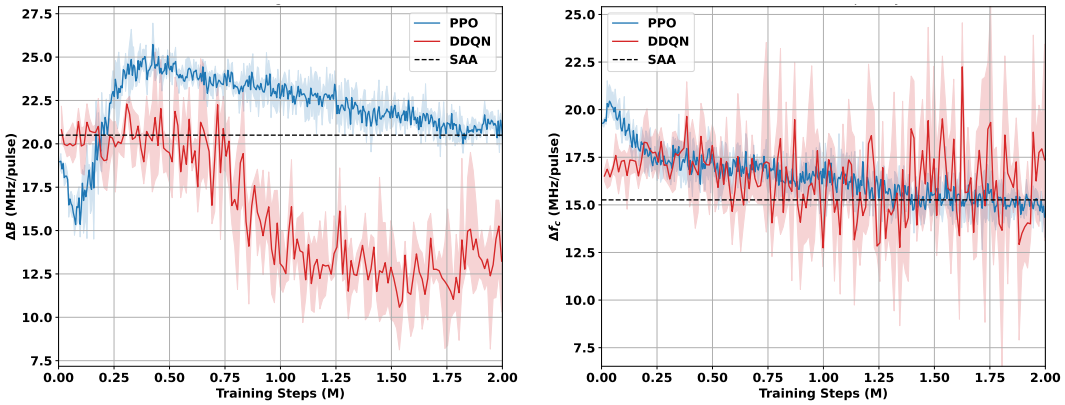


(b) Mean collision bandwidth



(c) Mean missed opportunity bandwidth

Figure 3.6: Spectrum sharing metrics in the 2.4-2.5 GHz ISM band (no waveform adaptation penalty).

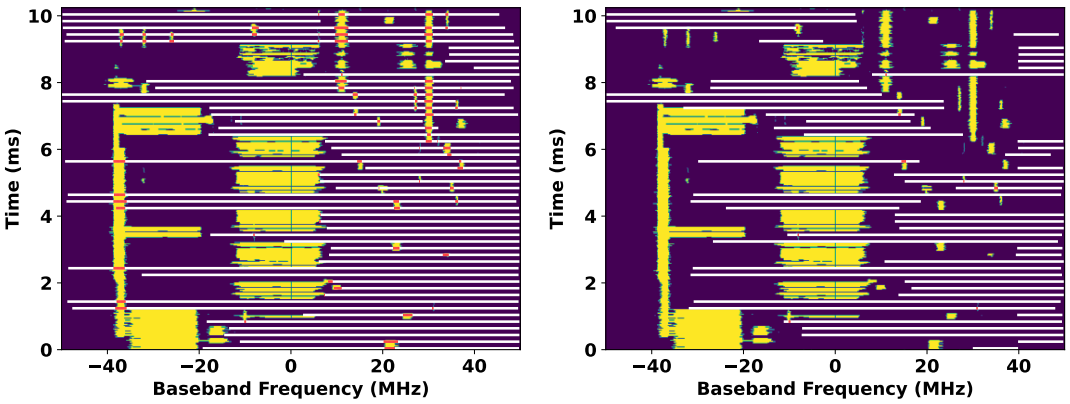


(a) Mean change in waveform bandwidth (b) Mean change in center frequency

Figure 3.7: Waveform adaptation metrics in the 2.4-2.5 GHz ISM band (no adaptation penalty).

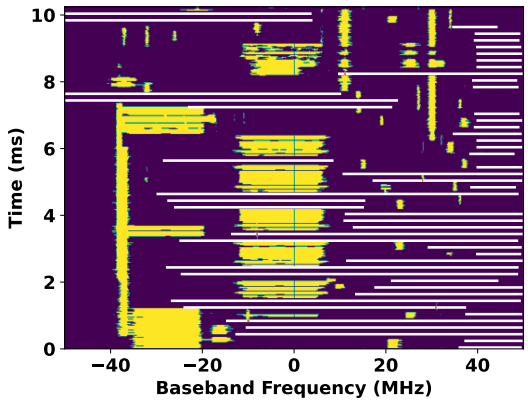
trum (Figure 3.6c). The discrete action space of the DDQN agent is the primary cause of this sub-optimal behavior. Since the DDQN agent can only operate in a small number of discrete frequency bands, it is unable to precisely position its waveform to exploit available openings in the spectrum. For example, if a wideband interference source lies between two bins in the DDQN agent's action space, the agent must either interfere with the user or avoid both bands entirely. The PPO agent avoids such issues by operating in a continuous action space.

Figure 3.7 shows the average change in bandwidth and center frequency per pulse during training. Since the waveform adaptation penalty from Equation (3.3) is disabled in this experiment, the agents are free to modify their waveforms without restriction. As in Figure 3.6, the PPO agent closely matches the behavior of the SAA agent. The DDQN agent adapts its waveform bandwidth less than the other two agents even without the penalty (likely because the DDQN agent utilizes less bandwidth overall). As will be shown in later experiments, adapting the waveform to this degree on each pulse leads to heavy degradation in pulse-Doppler radar image quality.



(a) $\alpha_c = 10$

(b) $\alpha_c = 30$



(c) $\alpha_c = 50$

Figure 3.8: Continuous control agent behavior for varying values of the collision penalty weight α_c. (a) When the penalty is small, the agent prioritizes bandwidth utilization and collides with many other users. (b) Moderate collision penalties encourage balanced behavior similar to a SAA policy while (c) large penalties encourage the agent to prioritize collision avoidance.

The spectrum sharing behavior of the agent is highly dependent on the collision penalty weight α_c in Equation (3.1). As discussed previously, the agent should aggressively pursue open spectrum when α_c is small and act conservatively to avoid interference when α_c is large. Figure 3.8 shows a representative example of three agent configurations in the same spectrum environment. The first agent is trained with a low collision penalty of $\alpha_c = 10$ (Figure 3.8a). Though the agent avoids wideband interference sources, it frequently collides with narrowband emitters in order to increase its waveform bandwidth. When the penalty is increased to $\alpha_c = 30$ (Figure 3.8b), the agent's behavior more closely approximates a SAA strategy as seen in the previous experiment. In this configuration, the agent avoids most persistent emitters and leaves small gaps where the spectrum was recently occupied. The agent becomes overly conservative when the penalty is further increased to $\alpha_c = 50$ (Figure 3.8c). In this case, the agent avoids nearly all other users but often leaves large swaths the spectrum unoccupied. Since $\alpha_c = 30$ most closely matches SAA behavior, it is used as a "default" value for spectrum sharing behavior.

Next, the behavior of the continuous control agent is evaluated in the 2.4-2.5 GHz band across various values of the distortion penalty from Equation 3.3. To balance the inherent trade-off between mutual interference mitigation and waveform adaptation, the reward function hyperparameters are constrained to be inversely related with $\beta = \beta_{bw} = \beta_{f_c}$ and $\alpha_c = 30 \cdot (1 - \beta)$. Formulated in this manner, the radar obtains a smooth trade-off between sense-and-avoid operation when β is low and full-band operation as β tends to unity.

Figure 3.9 shows the average training reward across several distortion values for the PPO agent. For all values of β aside from $\beta = 0$, the agent converges with similar stability and sample efficiency as in Figure 3.5. The $\beta = 0$ agent

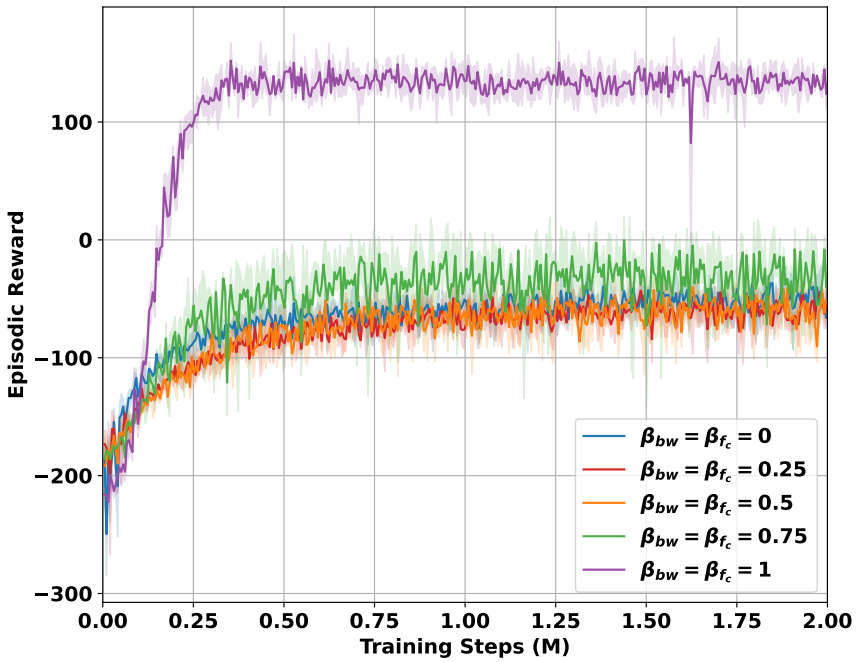
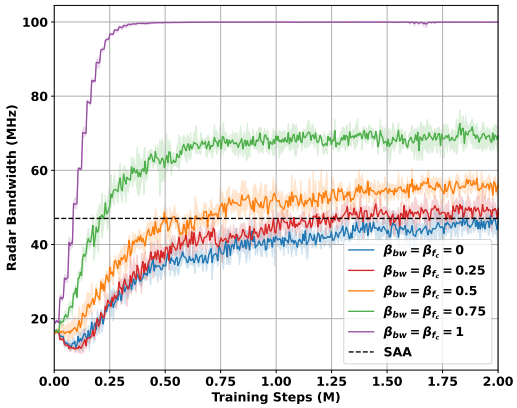


Figure 3.9: Episodic training reward in the 2.4-2.5 GHz ISM band (varying adaptation penalty).

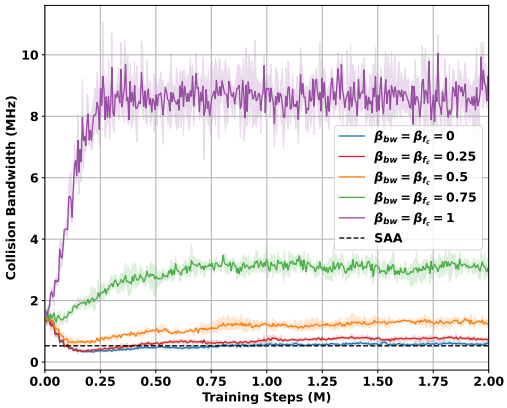
converges to a large reward because its optimal policy is to deterministically occupy the entire channel bandwidth. Overall, reasonable values of β do not impact the learning process.

Figure 3.10 quantifies the spectrum sharing behavior of the PPO agent as a function of the distortion coefficients β_{bw} and β_{f_c}. Large β values monotonically increase the agent's tolerance for mutual interference and, interestingly, its mean bandwidth utilization (Figure 3.10a). All but the $\beta = 0$ agent consistently utilize more bandwidth than the SAA agent. For $\beta \leq 0.5$, this results in only a slight increase in collision bandwidth (Figure 3.10b). However, increasing β further causes the agent to aggressively increase its bandwidth and collision rate. In these cases the missed opportunity bandwidth becomes negative (Figure 3.10c) since the radar occupies more bandwidth than the widest open band.

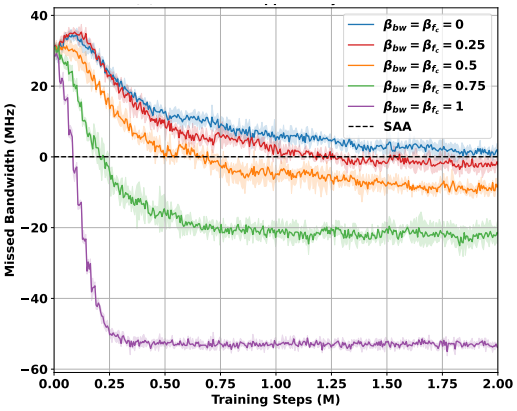
Figure 3.11 shows the impact of the β parameter on the agent's waveform



(a) Mean radar waveform bandwidth

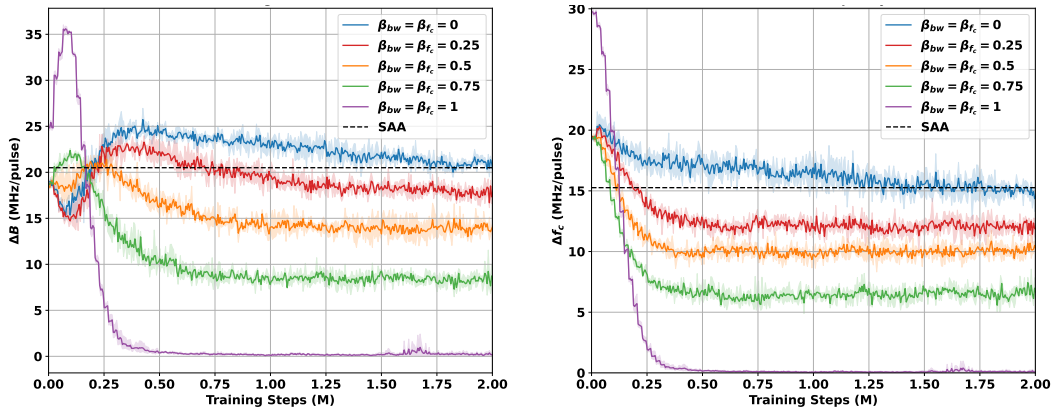


(b) Mean collision bandwidth



(c) Mean missed opportunity bandwidth

Figure 3.10: Spectrum sharing metrics in the 2.4-2.5 GHz ISM band (non-zero adaptation penalty).

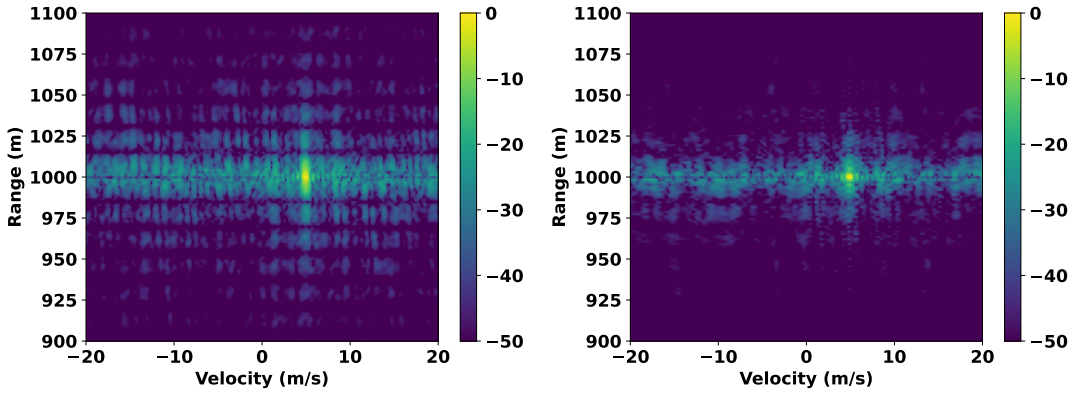


(a) Mean change in waveform bandwidth (b) Mean change in center frequency

Figure 3.11: Waveform adaptation metrics in the 2.4-2.5 GHz ISM band (non-zero adaptation penalty).

adaptation behavior. Larger β values consistently *decrease* the amount of pulse-to-pulse waveform adaptation within each CPI. Unlike in Figure 3.10, the reduction in adaptation is a more linear function of β. Therefore, the β parameter allows the agent to smoothly transition between a bandwidth-maximizing SAA strategy and a fixed emission radar. Although this experiment trained separate agents for each configuration, practical implementations could train one agent over a range of β values and adjust the parameter in real-time to meet operational requirements.

To relate the metrics in Figure 3.11 to radar performance, Figure 3.12 shows several range-Doppler point spread functions for varying values of β. When $\beta = 0$, the agent mimics the sense-and-avoid algorithm and makes drastic waveform modifications on each pulse, producing a range-Doppler map that is heavily distorted by RSM (Figure 3.12a). The RSM effect is appreciably reduced by increasing β to 0.5, although there is still smearing across Doppler. Setting β to 0.75 further reduces the distortion in the response, and the waveform's sidelobe structure is largely recovered. Finally, $\beta = 1$ (Figure 3.12d) corresponds to full-



(a) $\beta_{bw} = \beta_{f_c} = 0$

(b) $\beta_{bw} = \beta_{f_c} = 0.5$

(c) $\beta_{bw} = \beta_{f_c} = 0.75$

(d) $\beta_{bw} = \beta_{f_c} = 1$

Figure 3.12: Range-Doppler responses across various adaptation penalty values. Each response is representative of a typical 256-pulse CPI in the 2.4-2.5 GHz band. To balance the inherent trade-off between waveform adaptation and collision avoidance, the reward function parameters for each agent are linearly related with $\alpha_c = 30 \cdot (1 - \beta)$ and $\beta = \beta_{bw} = \beta_{f_c}$.

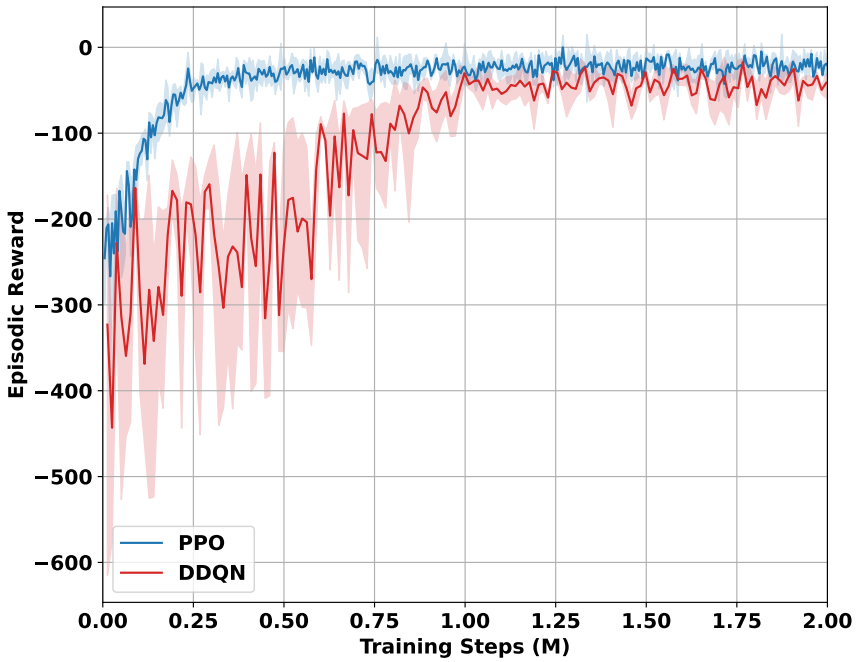


Figure 3.13: Episodic training reward in the 2.59-2.69 GHz band (no waveform adaptation penalty).

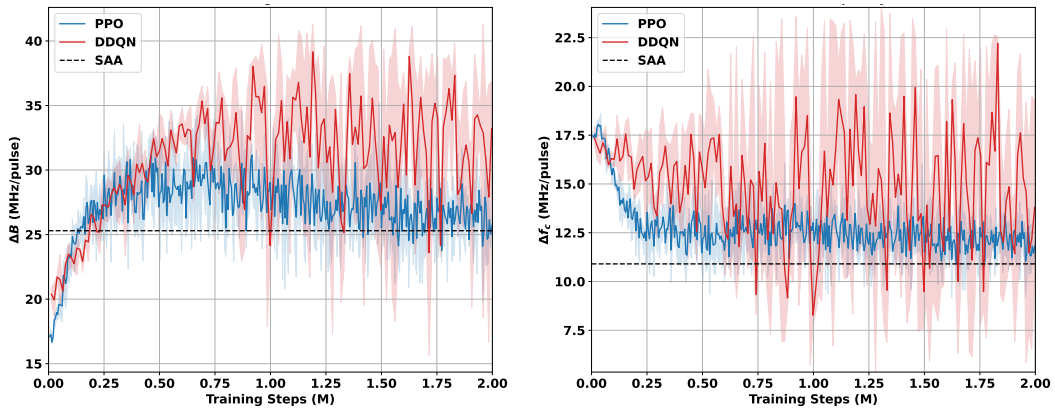
band operation, which recovers the ideal range-Doppler response with no RSM. Thus, the distortion penalty introduced in Equation 3.3 provides additional degrees of freedom for balancing the trade-off between collision avoidance and RSM mitigation.

3.3.2 2.59-2.69 GHz Environment

This section repeats the experiments from Section 3.3.1 in the 2.59-2.69 GHz band that is occupied exclusively by a cellular tower located next to the University of Oklahoma's Advanced Radar Research Center (ARRC). Since this band has only one primary user, its occupancy statistics are much more predictable over time than the 2.4-2.5 GHz ISM band. Although the 2.59-2.69 GHz band is unoccupied for long periods of time, the primary user occupies nearly the entire

channel when present. Overall, the band is significantly less dynamic and has rich temporal structure that a cognitive radar can learn.

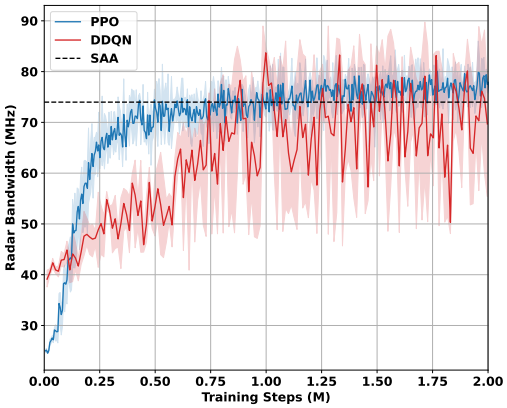
Figures 3.13-3.15 summarize the training dynamics of the agents in the 2.59-2.69 GHz band when only the spectrum reward in Equation (3.1) is optimized. Despite significant differences in the channel statistics, the trends from the previous section still largely hold. Figure 3.13 shows that the RL agent consistently obtains more reward than DDQN and once again converges in approximately half number the environment interactions.



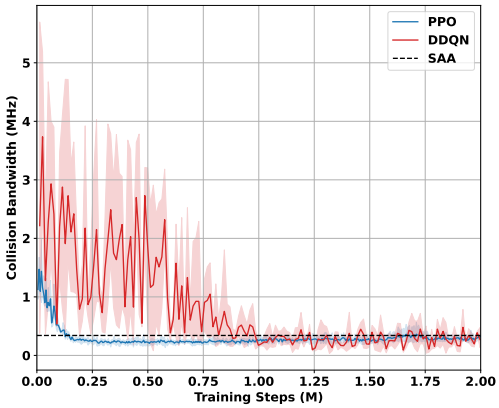
(a) Mean change in waveform bandwidth (b) Mean change in center frequency

Figure 3.15: Waveform adaptation metrics in the 2.59-2.69 GHz band (no waveform adaptation penalty).

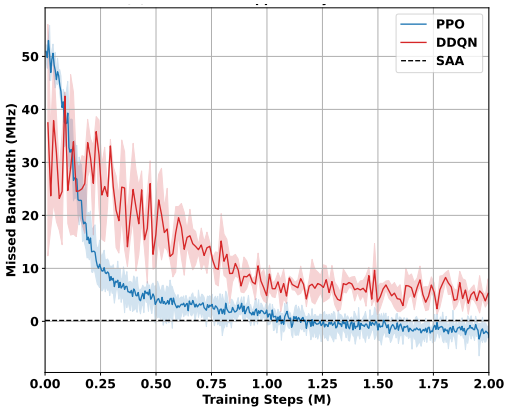
Both the PPO and DDQN agents exploit the temporal structure of the primary user to avoid mutual interference more successfully than the SAA agent (Figure 3.14b), and the PPO agent utilizes approximately 6-8 MHz more bandwidth than the DDQN on average (Figures 3.14a and 3.14c). Unlike for the 2.4-2.5 GHz experiment, the PPO agent utilizes 2-3 MHz more bandwidth than the SAA agent on average for the same collision bandwidth. This indicates that the predictive capabilities of the PPO agent are beneficial in this environment. Although none of the agents optimize for waveform adaptation, DDQN



(a) Mean radar waveform bandwidth



(b) Mean collision bandwidth



(c) Mean missed opportunity bandwidth

Figure 3.14: Spectrum sharing metrics in the 2.59-2.69 GHz band (no waveform adaptation penalty).

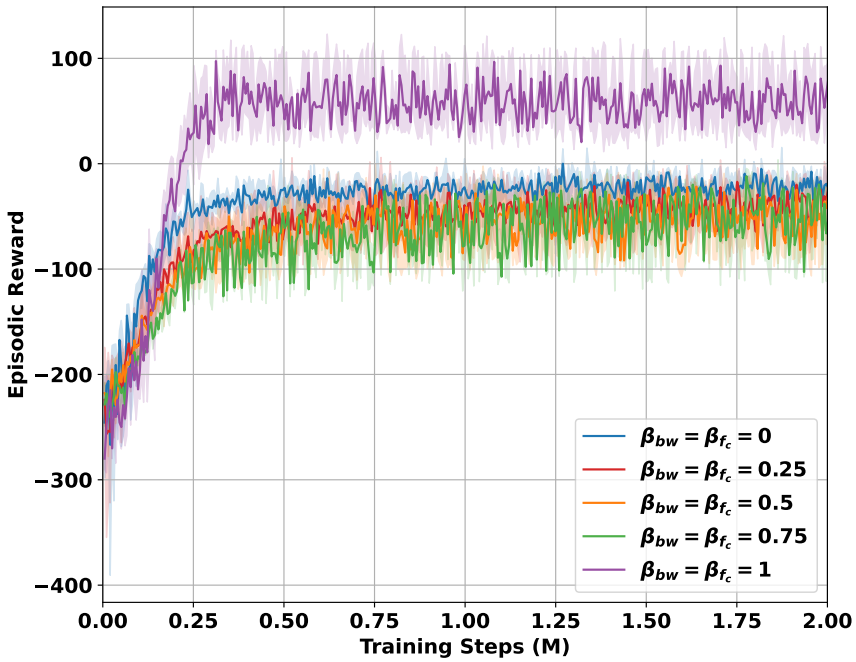
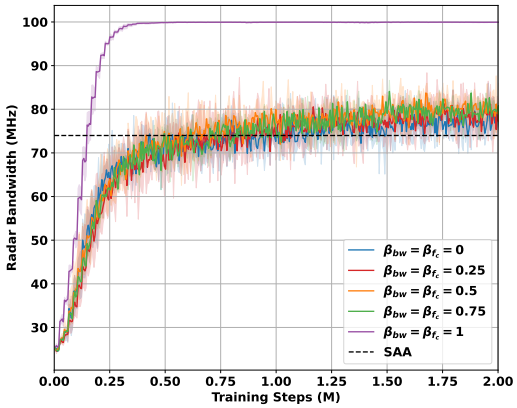


Figure 3.16: Episodic training reward in the 2.59-2.69 GHz band (varying adaptation penalty).

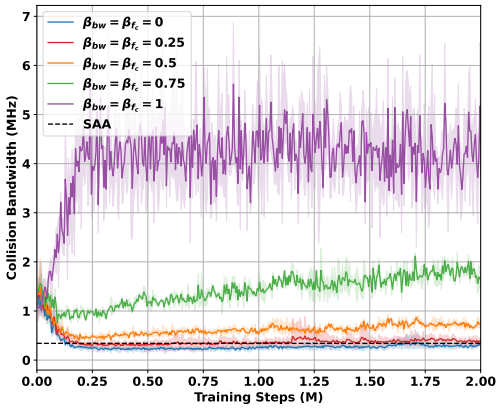
modifies its waveform parameters to a much greater extent than SAA or PPO (Figures 3.15a and 3.15b).

Figures 3.16-3.18b examine the trade-offs between bandwidth utilization and interference avoidance for varying adaptation parameter values in the 2.59-2.69 GHz band. As before, the reward parameters are linearly constrained as $\alpha_c = 30 \cdot (1 - \beta)$ with $\beta = \beta_{bw} = \beta_{f_c}$.

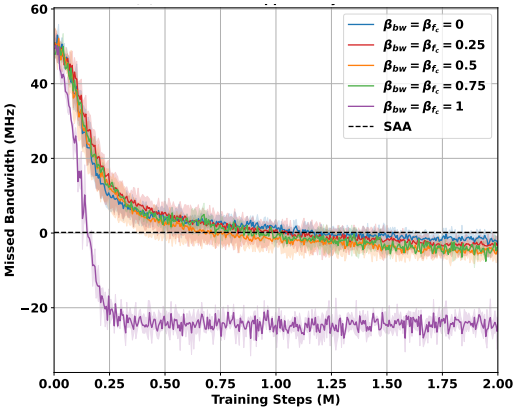
Again, the PPO agent consistently converges to a stable episodic reward after training for all values of β (figure 3.16). Unlike in the 2.4-2.5 GHz band, increasing the adaptation penalty causes the PPO agent to increase its interference tolerance *without* altering its bandwidth utilization for $\beta < 1$ (Figures 3.17a and 3.17b). This behavior arises because the primary user occupies nearly the entire channel when present, so the agent must choose to use the full bandwidth and collide with the primary user or avoid the primary user entirely by making



(a) mean radar waveform bandwidth

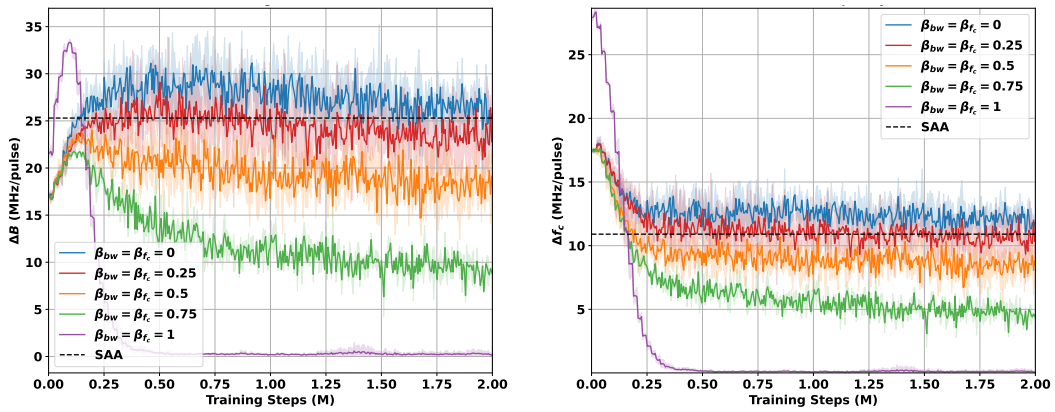


(b) Mean collision bandwidth



(c) Mean missed opportunity bandwidth

Figure 3.17: Spectrum sharing metrics in the 2.59-2.69 GHz band (varying waveform adaptation penalty).



(a) Mean change in waveform bandwidth (b) Mean change in center frequency

Figure 3.18: Waveform adaptation metrics in the 2.59-2.69 GHz band (varying waveform adaptation penalty).

excessively large changes to its emissions. The magnitude of these modification still monotonically decrease with increasing β, but the difference for small penalty values is less pronounced than in the previous experiment.

3.4 Summary

This chapter presented a novel deep RL technique for efficient RF spectrum access with a pulse-agile cognitive radar. Unlike existing RL approaches, this work formulates the waveform parameter optimization task as a continuous control problem where the agent selects the frequency-domain characteristics of its emissions from a continuous set of values. This allows the agent to precisely specify the frequency content of its emissions and scale its behavior to wideband, high-resolution operation that is not possible under prior learning methods. The proposed RL agent also uses a recurrent-attention encoder to process channel estimates at the full temporal and frequency resolution of the spectrum sensing process.

Next, a novel reward function was presented for cognitive radar spectrum sharing applications. The reward function simultaneously optimizes the radar's bandwidth utilization, mutual interference avoidance, and waveform adaptation behavior in high-dimensional spectrum environments. To fully characterize the properties of the resulting RL agent, several experiments were conducted using over-the-air data collected from the 2.4-2.5 GHz and 2.59-2.69 GHz bands using an SDR testing platform. These experiments demonstrate that the proposed technique is performant compared to baselines from the literature. They also show that the reward parameters can be tuned to produce a range of emergent agent behaviors. Although separate agents were trained for each reward configuration under consideration, it is also possible to train generalist agents that can adapt their behavior during inference by using several different reward configurations during training.

Chapter 4

Graph Learning for Autonomous Search and Track

This chapter analyzes sensor control techniques for the joint multi-object search and track problem. In search and track applications, an autonomous sensing platform must continuously search a region of interest for unknown objects while maintaining accurate state estimates on previously detected objects. Existing solutions to this problem from the literature typically rely on myopic heuristics or computationally intensive online planning routines that scale poorly to complex, high-dimensional control spaces. To address these shortcomings, this chapter proposes a novel graph learning solution that can be applied to the general class of search and track tasks with minimal scenario-specific tuning. This chapter is a significantly extended version of the preliminary conference publication [117] and much of the discussion that follows is adapted from the journal publication that is currently under review [118].

4.1 Problem Formulation

This section formulates the joint search and track problem as a partially-observable Markov decision process (POMDP) that can be solved using reinforcement learning. First, a graph representation for multi-object search and track scenarios is

Table 4.1: Node features

Feature	Agent	Search	Track
Class label	✓	✓	✓
Age (time steps)	✓	✓	✓
State vector	✓	✓	✓
Covariance diagonal	✓	✓	✓
Survival probability p_s	✓	✓	✓
Search weight		✓	
Sensor state	✓		
Track quality			✓
Existence probability r			✓

presented. The graph representation encodes the history of the environment as the scenario evolves over time. As will be shown, the graph structure can be constructed directly from the output of an RFS multi-object tracker and can be processed efficiently using graph neural networks (GNNs). Section 4.1.2 presents a performant GNN architecture for processing the scenario graphs. Due to the inherent sparsity of the problem and its graph representation, the time complexity of the graph encoder is approximately linear with respect to the number of entities in the environment. Since the TOMB/P filter can also perform data association in linear time (see Section 2.4.3), the overall pipeline has approximately linear time complexity and is well-suited for real-time operation. Section 4.1.3 derives a parameterized reward function that jointly optimizes search and track performance. The graph representation, encoder, and reward function have all been significantly streamlined compared to the initial conference publication [117].

4.1.1 Graph State Representation

The search and track agent models the environment state as a heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, U)$ which encodes the dynamics of the system as the active

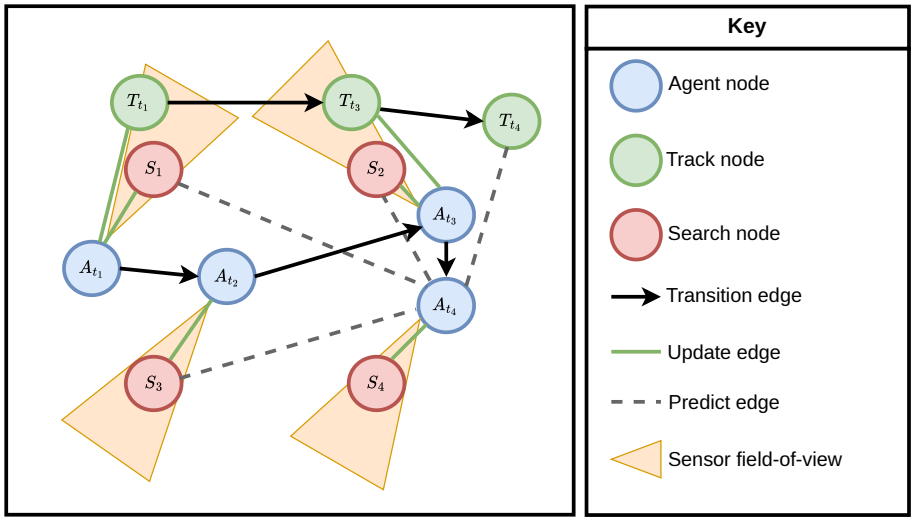


Figure 4.1: Representative illustration of the graph state representation. The scenario graph is composed of agent, search, and track nodes, where each node represents the state of an entity at a particular time. Entity nodes are connected via state transition, update, or predict edges.

sensor interacts with its environment over time. The set of nodes $\mathcal{V}$ includes three classes of entities: agent nodes, track nodes, and search nodes. Each edge in the set $\mathcal{E}$ represents a state transition over time or a measurement prediction/update between an agent and object. The global feature vector U describes the overall state of the multi-object tracker and scenario. Each component of the graph structure is described in turn, including the node/edge/global features and the relational rules which govern the graph topology.

The search and track problem involves three distinct types of entities/nodes. Agent nodes represent the sensing platform at each time step. The graph adds one agent node per time step, and each agent node connects to the previous agent node via a transition edge. Track nodes represent the state history of each tracked object. The graph adds one track node per tracked object at each time step and connects successive track states via transition edges. A measurement

update edge connects the track node to the corresponding agent node if the object was detected. Otherwise, the track node connects to the current agent node via a prediction edge. Search nodes represent the Gaussian mixture components of the TOMB/P filter undetected intensity described in Section 2.4.3. Since the search intensity may have an excessively large number of components, each component only retains its most recent state estimate node.

Figure 4.1 shows a stylized illustration of the proposed graph representation for a mobile beam-agile sensor. Nodes and edges are added incrementally to the graph as the scenario evolves over time. Due to the prediction edges, the current agent node can access every other node in the graph in at most three hops. This property ensures that the graph encoder network presented in Section 4.1.2 requires no more than three GNN layers to access the complete graph state.

Each node also contains a feature vector describing its corresponding entity (Table 4.1). All node types share several common features, including a one-hot class label, the node age (in time steps), and kinematic information such as the state vector and covariance matrix. Agent nodes contain a feature vector describing the internal sensor state. For instance, a beam-agile radar sensor state may consist of the current steering angle and beamwidth while a mobile platform state may include its heading and speed. Track nodes contain features describing the track's state estimate, such as its track quality and existence probability. Search nodes contain the weight of their corresponding intensity component. In contrast to the preliminary work in [117], the graph encoder shares a unified feature space across all node types. Features that are not used for a given node type (i.e., with no ✓in Table 4.1) are filled with zeros.

The graph models three different relation types between entities (Table 4.2). As for the node features, edges share a common feature space. Each edge con-

Table 4.2: Edge features

Feature	Transition	Update	Predict
Class label	✓	✓	✓
Relative pose	✓		
Detection probability		✓	
Measurement vector		✓	✓

tains a one-hot label vector indicating its class type. State transition edges encode relative pose information (e.g., change in position and orientation) between subsequent time steps. Both prediction and update edges contain the measurement vector that would be produced from a noise-free measurement of the state estimate mean. Measurement update edges also contain the detection probability.

The graph contains a 5-dimensional global feature vector that describes the overall tracker state. The global features are listed below:

- The sum of undetected intensity component weights
- The number of active tracks
- The sum of all track scores (Equation (4.2))
- The search task priority c_s (Equation (4.1))
- The track task priority c_t (Equation (4.1))

Since the scale of several node, edge, and global features may vary significantly depending on the scenario, the encoder applies a symmetric logarithm (symlog) transformation to all features prior to processing [83]. This eliminates the need for manual feature normalization across different scenarios and prevents gradient explosion due to large inputs.

4.1.2 Graph Encoder Network

This section presents a neural network architecture for processing the environment graphs discussed in Section 4.1.1. The graph encoder maps a variable-size input graph to a fixed-dimension latent vector suitable for downstream learning tasks. Figure 4.2 outlines the network topology. The network executes three processing stages: encoding, graph processing (relational reasoning), and a final decoding step.

The encoding stage independently projects each node, edge, and global feature vector into a shared latent space using a single dense layer. Unlike the preliminary work in [117], the graph is homogeneous and the parameters of the projection layers are shared across all node and edge types. This reduces the overall number of learnable parameters and improves generalization across different scenarios. The encoding step also projects the global features into the latent space and adds the result to each node and edge embedding to provide global context during graph processing. Regardless of the GNN type, skip connections are applied to mitigate oversmoothing problems that are common in GNNs [119]. RMSNorm [120] is also applied prior to each GNN layer to improve training stability.

The final decoding step distills the latent graph state into a single feature vector. Many pooling operators can be used to aggregate information across graphs and sets [121], [122]. This work uses the embedding vector of the current agent node to represent the entire graph. The current agent node is highly central due to the prediction and update edges connecting it to all other entities in the scenario. Therefore, it encodes both local ego information about the sensor state and global structural information across the graph. Additionally, directly using the agent node embeddings avoids the need for potentially expensive pooling

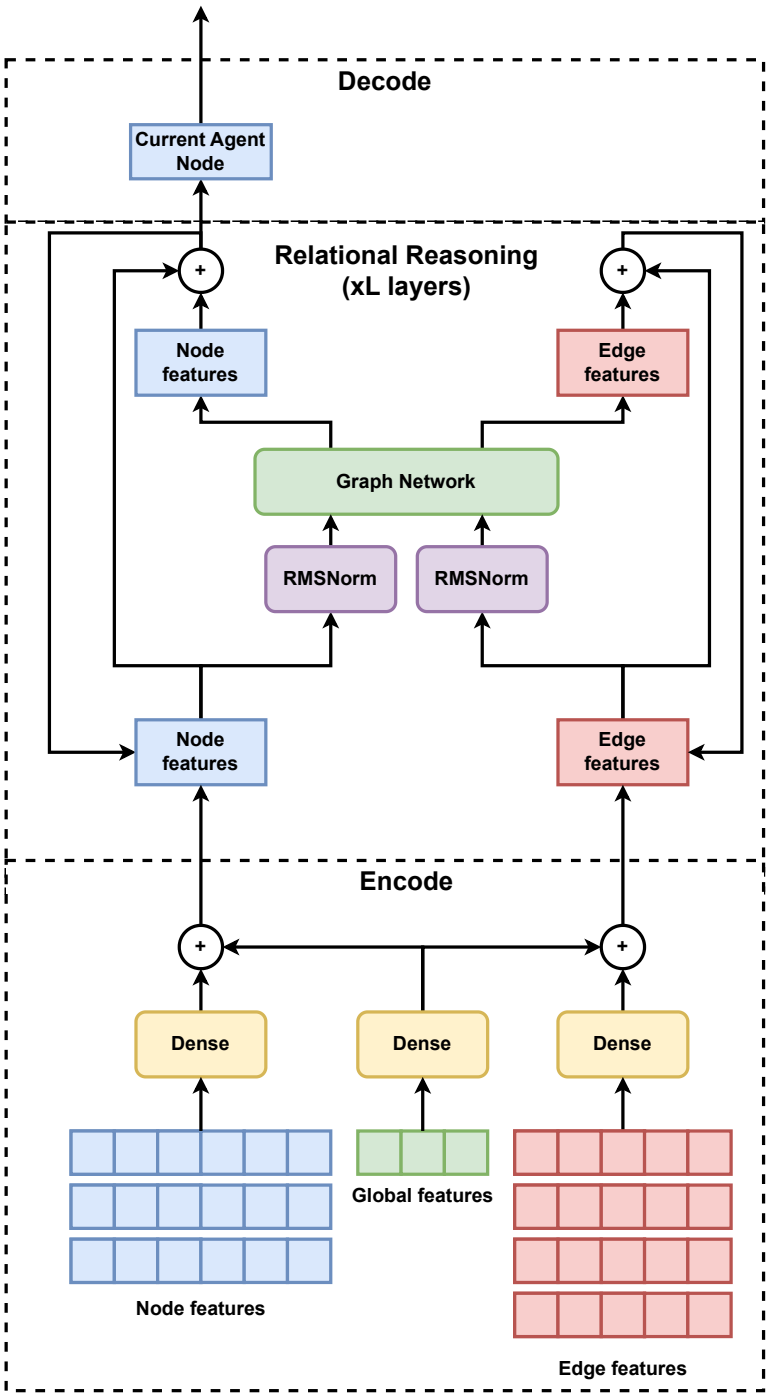


Figure 4.2: Graph processing network architecture. The network first projects each node, edge, and global feature vector into a shared latent space. Next, the model forms a graph and performs L layers of GNN processing on the entire graph. The decoding stage extracts the latent representation of the agent node from the current timestep to represent the complete graph state.

operations. Given the output latent vector, the graph encoder uses a dense layer to project the embedding into the TD-MPC2 world model latent space.

4.1.3 Reward Function

Given the graph state representation and encoder network, all that remains is to define a suitable reward function that guides the RL agent towards optimal search and track behavior. The total reward R_t at time t is a weighted combination of individual task rewards:

$$R_t = c_s R_t^{search} + c_t R_t^{track}, \tag{4.1}$$

where R_t^{search} and R_t^{track} are the search and track reward components, respectively, and $c_s, c_t \geq 0$ are the priority weights for each task. Ideally, this reward function should drive the agent to maintain high-quality state estimates on as many objects as possible while devoting sufficient resources to discovering new objects in the surveillance region. To minimize the amount of prior knowledge required during training, the reward components should also depend only on the multi-object filter state rather than ground truth information which may not be available during online training.

To evaluate tracking task performance, let $\rho(x^{(i)})$ be a score function that maps each track state estimate $x^{(i)}$ to a numerical score in the range $[0, 1]$, where higher scores indicate better track quality. The track reward defines the agent's overall tracking performance as the sum of individual scores across all active tracks:

$$R_t^{track} = \sum_{i \in \mathcal{C}} \rho\left(x_{t|t}^{(i)}\right), \tag{4.2}$$

where $\mathcal{C}$ is the set of "confirmed" tracks. A track is defined as confirmed if

its existence probability exceeds a user-specified threshold r_{th} and it has been associated to at least M measurements. These constraints encourage the agent to confirm tentative tracks and ensure the agent is not rewarded for false tracks. Note that the tracking reward is strictly non-negative and achieves its maximum value when all tracks have a score equal to unity.

Countless metrics can be used to quantify the track quality of a state estimate ([123] describes several metrics and their shortcomings). This work models each object state as a Gaussian distribution and uses the state covariance trace as an error measure. The track score for a state estimate $x^{(i)}$ with covariance $P^{(i)}$ is computed as:

$$\begin{aligned} \rho(x^{(i)}) &= \max\left(1 - \frac{tr(WP^{(i)}W^T)}{tr^{\max}}, 0\right) \\ &= \max\left(1 - e^{(i)}, 0\right), \end{aligned} \tag{4.3}$$

where $e^{(i)}$ is the trace error ratio in the first equality, $tr^{\max}$ is a user-specified maximum tolerable trace, and W is a weight matrix that extracts and scales specific matrix elements. Equation (4.3) assigns a score of unity to tracks with zero trace in the relevant dimensions, and the score decays linearly to zero as the trace approaches or exceeds $tr^{\max}$. For the experiments in Section 4.3, W extracts the position components of the covariance matrix without scaling.

For the search task, the agent must repeatedly explore the surveillance region to detect and localize as many objects as possible. As discussed in Section 2.4.3, the TOMB/P filter explicitly models undetected objects using a Poisson RFS with intensity function $\lambda^u(x)$. This intensity function describes the number of undetected objects per unit volume in the state space. Therefore, integrating the $\lambda^u(x)$ over the surveillance region yields the expected *total* number of undetected

objects. The search reward is defined to minimize this quantity directly:

$$R_t^{search} = -\int \lambda^u(x)dx = -\sum_{i=1}^{N_u} w_{t|t}^{(i)}. \tag{4.4}$$

The second equality in Equation (4.4) holds when the intensity function is represented as a Gaussian mixture of the form:

$$\lambda^u(x) = \sum_{i=1}^{N^u} w_u^{(i)} \mathcal{N}(x; \mu_u^{(i)}, P_u^{(i)}). \tag{4.5}$$

Equation (4.4) is a strictly non-positive penalty term that is only zero when the undetected intensity is uniformly zero (i.e., when the tracker predicts no undetected objects remain in the region).

Substituting Equations (4.4) and (4.2) into Equation (4.1) gives:

$$\begin{aligned} R_t &= c_s R_t^{search} + c_t R_t^{track} \\ &= -c_s \sum_{i=1}^{N^u} w_{t|t}^{(i)} + c_t \sum_{j \in \mathcal{C}} \rho(x_{t|t}^{(j)}) \\ &= -c_s \sum_{i=1}^{N^u} w_{t|t}^{(i)} + c_t \sum_{j \in \mathcal{C}} \max(1 - e, 0) \\ &= -c_s \sum_{i=1}^{N^u} w_{t|t}^{(i)} + c_t \sum_{j \in \mathcal{C}} 1 - c_t \sum_{j \in \mathcal{C}} \min\left(e^{(j)}, 1\right) \\ &= c_t N_{\text{confirmed}} - c_t \sum_{j \in \mathcal{C}} \min\left(e^{(j)}, 1\right) - c_s \sum_{i=1}^{N^u} w_{t|t}^{(i)}. \end{aligned} \tag{4.6}$$

Therefore, the search and track error decomposes into three distinct terms. The first term in the last equality rewards the agent for *confirming* tentative tracks. The second term is a penalty proportional to the state estimation error in each track. This term must be clipped to ensure that the overall reward cannot decrease by confirming a track. The third term is simply the search penalty

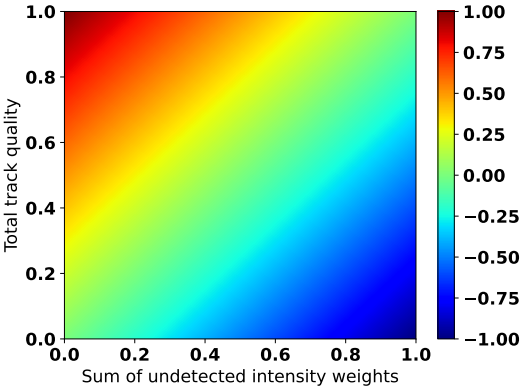
from Equation (4.4), which gives the expected number of undetected objects that remain in the surveillance region.

Figure 4.3 shows the reward function behavior for several task priority values as the track quality and sum of undetected intensity weights vary from 0 to 1. Warm and cool colors indicate regions of high and low reward, respectively. When the task priorities are equal, the pareto fronts for each reward region lie along diagonal lines in this space (Figure 4.3a). Increasing the relative priority of the search task decreases the slope of these pareto optimal regions (Figures 4.3b and 4.3d), indicating that the reward becomes less sensitive to changes in the track quality. Conversely, increasing the relative tracking task priority drives the optimal regions to lie between increasingly vertical lines in the space (Figures 4.3c and 4.3e).

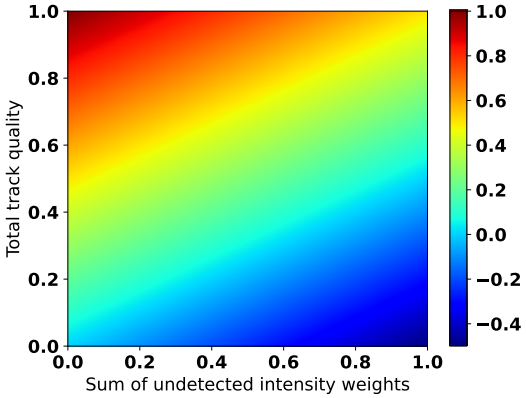
4.2 Experiment Setup

4.2.1 Baseline Comparisons

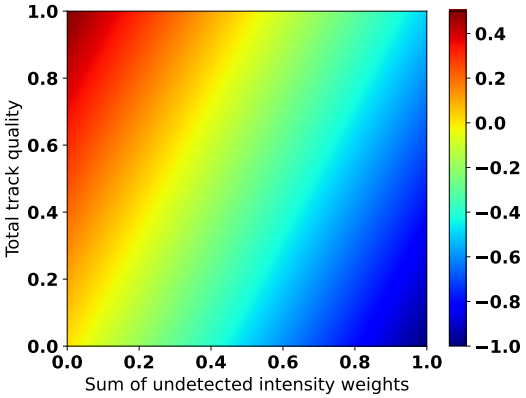
Section 4.3 compares the proposed graph RL approach to the sensor control technique from [42]. The authors of [42] formulate the joint search and track task as an optimal control problem and solve it via explicit planning in an environment simulator. When the number of feasible control inputs is small, the planning approach simulates all possible action sequences and selects the action with the lowest expected cost. This exhaustive search strategy becomes computationally infeasible for larger action spaces and longer planning horizons, so the authors also introduce a Monte Carlo Tree Search (MCTS) technique to select control actions [43]. Both planners select actions in receding-horizon fashion, re-planning the entire action sequence at each time step as new measurements arrive at the



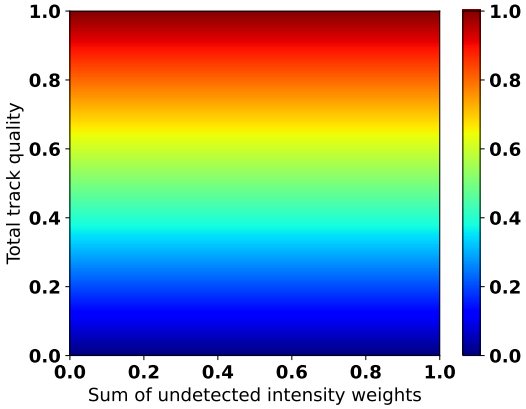
(a) $c_s = c_t = 1$



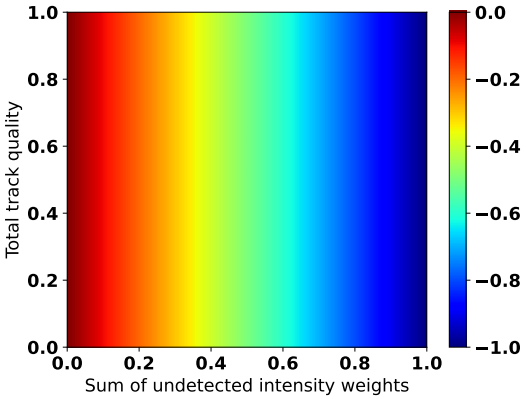
(b) $c_s = 0.5,\ c_t = 1$



(c) $c_s = 1,\ c_t = 0.5$



(d) $c_s = 0,\ c_t = 1$



(e) $c_s = 1,\ c_t = 0$

Figure 4.3: Reward heatmaps over several values of the task priority weights. Each image shows the reward landscape as a function of the undetected intensity weight (x-axis) and total track quality (y-axis). Red indicates high-reward regions while blue indicates low-reward regions.

sensor.

Unlike this work, the planners from [42] use the Poisson Multi-Bernoulli Mixture (PMBM) filter from [18] as the underlying multi-object tracking filter. The PMBM filter maintains an explicit hypothesis tree for data association and requires computationally expensive data association techniques to update the multi-object state. To avoid this computation during planning, the planners assume that each object in the sensor field-of-view produces a noise-free measurement with no false alarms or new object births. To improve robustness to false alarms, the planning procedure only considers tracks with existence probability $r > r_{\text{th}}$, where r_{th} is set to a small value to encourage the agent to confirm or delete low-confidence tracks.

4.2.2 Metrics

The experiments in Section 4.3 characterize the performance of each autonomous agent in terms of its search and track capabilities, runtime computational requirements, and learning efficiency. The generalized optimal sub-pattern assignment (GOSPA) metric from [124] is a common metric for quantifying joint search and track performance. In its most general form, GOSPA measures the distance between two finite sets X and Y as:

$$\begin{aligned} & d_p^{(c,\alpha)}(X,Y) \\ & = \left(\min_{\pi \in \prod_{|Y|}} \sum_{i=1}^{|X|} d^{(c)}\left(x_i, y_{\pi(i)}\right)^p + \frac{c^p}{\alpha}\left(|Y| - |X|\right) \right)^{\frac{1}{p}}, \end{aligned} \tag{4.7}$$

where $d^{(c)}(x,y) = \min(d(x,y), c)$ for some distance metric $d(\cdot,\cdot)$ and cut-off value $c > 0$, $\prod_{|Y|}$ is a set containing all possible permutations of Y, $1 \leq p < \infty$ controls the sensitivity to outlier values, and $\alpha \leq 2$.

The authors of [124] suggest the constraint $\alpha = 2$ for multi-object tracking applications, which simplifies Equation (4.7) to:

$$\begin{aligned} & d_p^{(c,2)}(X,Y) \\ & = \min_{\gamma_a \in \Gamma} \left(\sum_{(i,j)\in\gamma_a} d\left(x_i, y_j\right)^p + \frac{c^p}{2}\left(|X| + |Y| - 2|\gamma_a|\right) \right)^{\frac{1}{p}}, \end{aligned} \tag{4.8}$$

where Γ is the set of all one-to-one assignments between the elements of X and Y. Equation (4.8) decomposes into components that independently measure track localization errors, false track errors, and missed track errors. These components are respectively defined as:

$$d_{loc}(X,Y,\gamma_a^\star) = \sum_{(i,j)\in\gamma_a^\star} d\left(x_i, y_j\right)^p, \tag{4.9}$$

$$d_{false}(X,Y,\gamma_a^\star) = \frac{c^p}{2}\left(|X| - |\gamma_a^\star|\right), \text{ and} \tag{4.10}$$

$$d_{miss}(X,Y,\gamma_a^\star) = \frac{c^p}{2}\left(|Y| - |\gamma_a^\star|\right), \tag{4.11}$$

where $\gamma_a^\star$ is the assignment that minimizes Equation (4.8) for a multi-object state estimate X and ground truth set Y.

In other words, objects that are associated to ground truth in the optimal assignment $\gamma_a^\star$ contribute to the localization error in Equation (4.9). Objects in X that are not assigned to a ground truth element contribute to the false error in Equation (4.10), increasing the GOSPA error by $c^p/2$ for each false track. Similarly, ground truth objects in Y that are not associated to a track contribute $c^p/2$ to the GOSPA via the miss error in Equation (4.11). The analysis in this chapter computes the optimal assignment $\gamma_a^\star$ using the modified Jonker-Volgenant algorithm derived in [125], [126]. In addition to Equations (4.9)-(4.11),

the experiments also report the sum of undetected intensity weights and total track quality as measures of search and track performance.

Runtime complexity is also a critical consideration for autonomous sensor control techniques. This chapter evaluates the runtime complexity and scalability of the planning agents from [42] as a function of the planning horizon and size of the control input space. Since the runtime of the graph RL agent does not scale appreciably with these quantities, its computational efficiency is characterized as a function of the scenario graph size. Finally, the learning efficiency of the graph RL agent is reported by plotting its learning curves in each scenario over 20 random seeds.

4.2.3 Assumptions and Common Parameters

The simulation studies in Section 4.3 use the TOMB/P filter described in Section 2.4.3 for multi-object state estimation and an unscented Kalman filter (UKF) for single-object state estimation. The UKF generates sigma points as described in [101] with parameters $\alpha = 10^{-3}$, $\beta = 2$, and $\kappa = 0$. Although the planning baselines from [42] use a PMBM filter for multi-object state estimation and an extended Kalman filter (EKF) for single-object state estimation, they use the TOMB/P filter and UKF in these experiments to ensure a fair comparison of state estimation performance. Both the graph RL and planning agents use an existence probability threshold of $r_{th} = 0.05$ to define active/confirmed tracks for track maintenance and reward computation.

The TOMB/P filter implementation models the birth and undetected intensities as Gaussian mixtures with $N^{(b)}$ and $N^{(u)}$ components, respectively. Birth components added to the undetected intensity in the predict step are merged after the update step to maintain a constant number of undetected intensity

components. Classical mixture reduction techniques (e.g., [127]) are computationally expensive, so each birth component is simply merged into the nearest undetected component in terms of Euclidean distance between means. To further reduce simulation complexity, detection and survival probabilities are evaluated using only the mean of the object state density:

$$p_d(x^{(i)}|s) \approx p_d(\mathbb{E}[x^{(i)}]|s), \text{ and} \tag{4.12}$$

$$p_s(x^{(i)}) \approx p_s(\mathbb{E}[x^{(i)}]|s) \tag{4.13}$$

where s is the sensor state and $x^{(i)}$ is the state of object i. Additionally, the sensor models in the experiments that follow are intentionally made simple and general to capture the general behavior of many common sensor types (e.g., radar, lidar, camera) without being tied to a specific application.

The RL agent uses the graph encoder architecture described in Section 4.1.2 to process the environment graph at each time step. The relational reasoning module uses $L = 3$ residual gated graph ConvNet (R-GGCN) layers with an embedding dimension of 128 for all node, edge, and global features. The resulting output is projected to a 512-dimensional latent vector using a single dense layer, then passed to the TD-MPC2 world model. The graph encoder has approximately $320k$ parameters in total. All other TD-MPC2 hyperparameters are identical to the default $5M$ parameter model from [81].

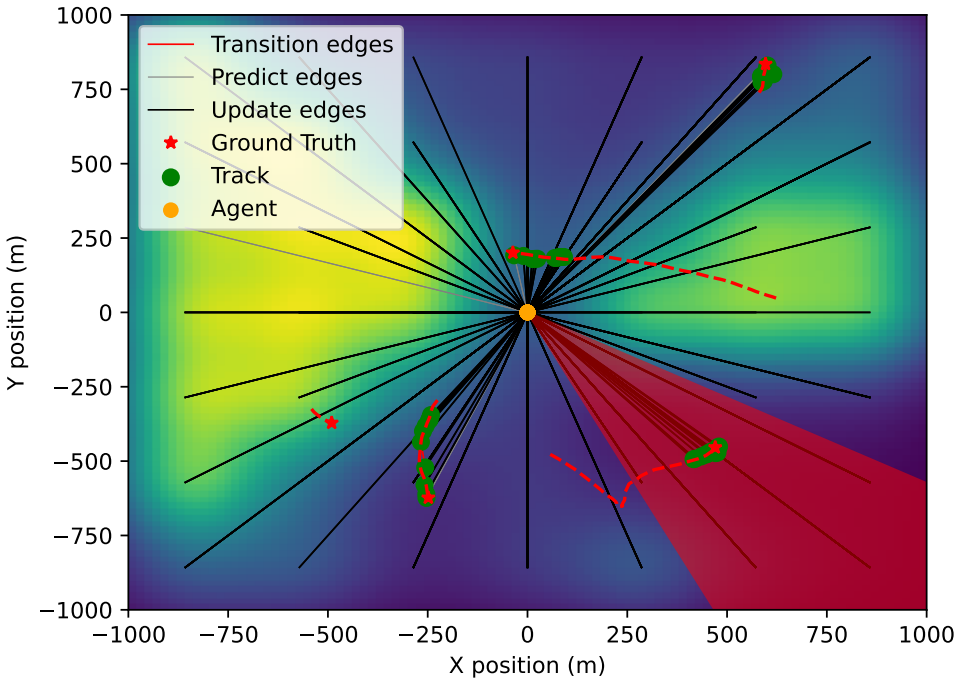


Figure 4.4: Example realization of the beam optimization scenario. The stationary sensing platform (yellow circle) controls an electronically-steered beam (red sector). In this instance, the sensor maintains 4 active tracks (green circles) and has yet to detect one ground truth object.

4.3 Results

4.3.1 Beam Optimization Task

The first experiment evaluates each autonomous agent in a 2-D scenario where a stationary sensing platform operates in a 2000 m × 2000 m region of interest (Figure 4.4). The sensor lies at the origin of the coordinate system and uses an electronically-steered beam to localize objects in the environment. At each decision step, the agent selects the steering angle θ_{steer} and beamwidth θ_{BW} of the beam.

The sensor's detection model is defined as:

$$p_d(x^{(i)}) = \begin{cases} 0.6 + 0.3 \cdot (\theta_{\text{BW}} - \theta_{\text{BW}}^{\min}) & \text{in beam,} \\ 0 & \text{otherwise,} \end{cases} \tag{4.14}$$

where $\theta_{\text{BW}}^{\min}$ is the minimum possible beamwidth of the sensor and an object with state $x^{(i)}$ is considered to be in the beam if its azimuth $\phi^{(i)}$ lies within the range $[\theta_{\text{steer}} - \theta_{\text{BW}}/2, \theta_{\text{steer}} + \theta_{\text{BW}}/2]$. In other words, the model assumes the detection probability is constant within the beam and zero outside, and that the probability decays linearly from 0.9 to 0.6 as the beamwidth increases to its maximum value. The sensor steers its beam instantaneously from $0^\circ - 360^\circ$ with beamwidth values from $20^\circ - 40^\circ$.

The active sensor measures the radial range and azimuth of objects in the environment. The measurement likelihood function is Gaussian distributed as $p(z_t|x_t) = \mathcal{N}(z_t; h(x_t), R_t)$ with mean and covariance defined below:

$$h(x_t) = \begin{bmatrix} \sqrt{p_x^2 + p_y^2} \\ \text{atan2}(p_y, p_x) \end{bmatrix}, \quad R_t = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\theta^2 \end{bmatrix}, \tag{4.15}$$

where (p_x, p_y) is the Cartesian position of the object relative to the sensor and σ_r, σ_θ are the measurement errors in range and azimuth, respectively. This experiment assumes that measurement errors are constant as a function of time and sensor state, although this need not be the case in general.

The kinematic state of each object includes its 2-D position (p_x, p_y) and velocity (v_x, v_y) to form a state vector $x = [p_x, v_x, p_y, v_y]$. Object states evolve according to a nearly constant velocity (NCV) model with transition density

$p(x_{t+1}|x_t) = \mathcal{N}(x_{t+1}; F_{CV}x_t, Q_{CV})$, where:

$$F_{CV} = I_2 \otimes \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, \quad Q_{CV} = qI_2 \otimes \begin{bmatrix} \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^2}{2} & \Delta t \end{bmatrix}. \tag{4.16}$$

Here, I_2 is the 2×2 identity matrix, $\otimes$ is a Kronecker product, Δt is the time difference between state transitions, and q is the process noise intensity. Objects enter the scenario according to a Poisson process which produces a new object every 50 s on average. When an object is born, its position is uniformly sampled within the scenario extents and its speed is uniformly selected between $0-10$ m/s with a random heading. Objects continue along their trajectories until they exit the scenario extents, at which point they are removed from the simulation.

The TOMB/P birth model is a Gaussian mixture of the form in Equation (4.5). To approximate a diffuse prior, the means of each component $\mu^{b,i} = \left[p_x^{b,i}, v_x^{b,i}, p_y^{b,i}, v_y^{b,i}\right]$ are selected such that $p_x^{b,i}$ and $p_y^{b,i}$ form a uniform 7×7 grid in position and $v_x^{b,i} = v_y^{b,i} = 0$. The mixture covariances are equal for all birth components and defined as $P^{b,i} = \text{diag}\left([2000/7, 10, 2000/7, 10]\right)^2$ to cover the region of the state space where objects may exist. The birth mixture weights are equal and set to $w^{b,i} = (1/50)/49 = 4.081 \times 10^{-4}$ for all components, corresponding to an expected object birth every 50 s. The survival probability is $p_s = 0.997$ within the surveillance region and zero outside. Table 4.3 summarizes the relevant parameters discussed in this section.

Since the platform is stationary and the beam is steered instantaneously, the the baseline planning agent from [42] need only optimize its behavior over a single time step. Given sufficiently fine discretization and an accurate environment simulator, the exhaustive planner is nearly optimal for the beam optimization task in terms of joint search and track performance. Here, the planner discretizes

Table 4.3: Beam Optimization Scenario Parameters

Parameter	Value	Unit
Scenario extents	2000×2000	m
Steering angle	$0 - 360$	degrees
Beamwidth	$20 - 40$	degrees
Max sensor range	1500	m
Max localization error	50	m
Δt	1	s
Range measurement error	5	m
Azimuth measurement error	1	degrees
Object motion process noise	0.1	—
Average birth rate	0.02	s^{-1}
Birth model	Uniform Poisson	—
Birth grid size	7×7	—
Survival probability	0.997	—
RL discount factor	0.95	—

the control space into 20 uniformly spaced steering angle bins and 5 beamwidth bins, resulting in 100 feasible control action. At each decision step, the planning routine explicitly simulates all possible actions and selects the action which maximizes the joint search+track cost function from [42]. Though they were not implemented here, this exhaustive search procedure also is similar to the approaches employed in [14].

Figure 4.5 shows the mean training reward of the graph RL agent in the beam optimization task, along with its 95% confidence interval. The dashed black line indicates the average reward of the exhaustive search planner over 100 Monte Carlo episodes with 1000 decision steps each. All agents assume equal priority between the search and track tasks and set $c_s = c_t = 1$ in Equation (4.1) during training. The RL agent converges to a similar episodic reward as the planning baseline after $2M$ environment steps, though there is significant variation in reward across seeds since environment realizations are heavily randomized.

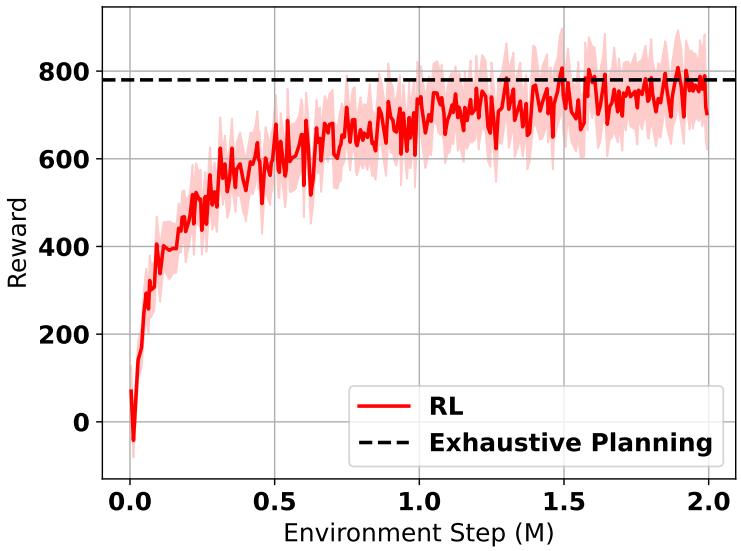


Figure 4.5: Learning curves for the beam optimization task over 20 random seeds. The black line gives the average reward of the MCTS planning agent from [42] over 100 Monte Carlo episodes. The RL agent obtains similar task performance as the planner after training is complete.

Figure 4.6 shows the relevant search and track performance metrics for the beam optimization task. This experiment compares the exhaustive search planner to three RL configurations: search+track (S+T) with $c_s = c_t = 1$ in Equation (4.1), search-only with $c_s = 1, c_t = 0$, and track-only with $c_s = 0, c_t = 1$. Since the S+T case is the primary focus of this work, metrics are computed from the average across the 20 S+T independently-trained RL agents with 100 Monte Carlo episodes per agent. The metrics for other configurations are averaged over 100 Monte Carlo episodes.

The S+T RL agent and planner perform best on the joint search and track task, achieving average GOSPA values of 38.068 and 37.671, respectively, compared to 48.525 for the search-only agent and 43.405 for the track-only agent (Figure 4.6a). To verify the statistical significance of this result, a two-tailed

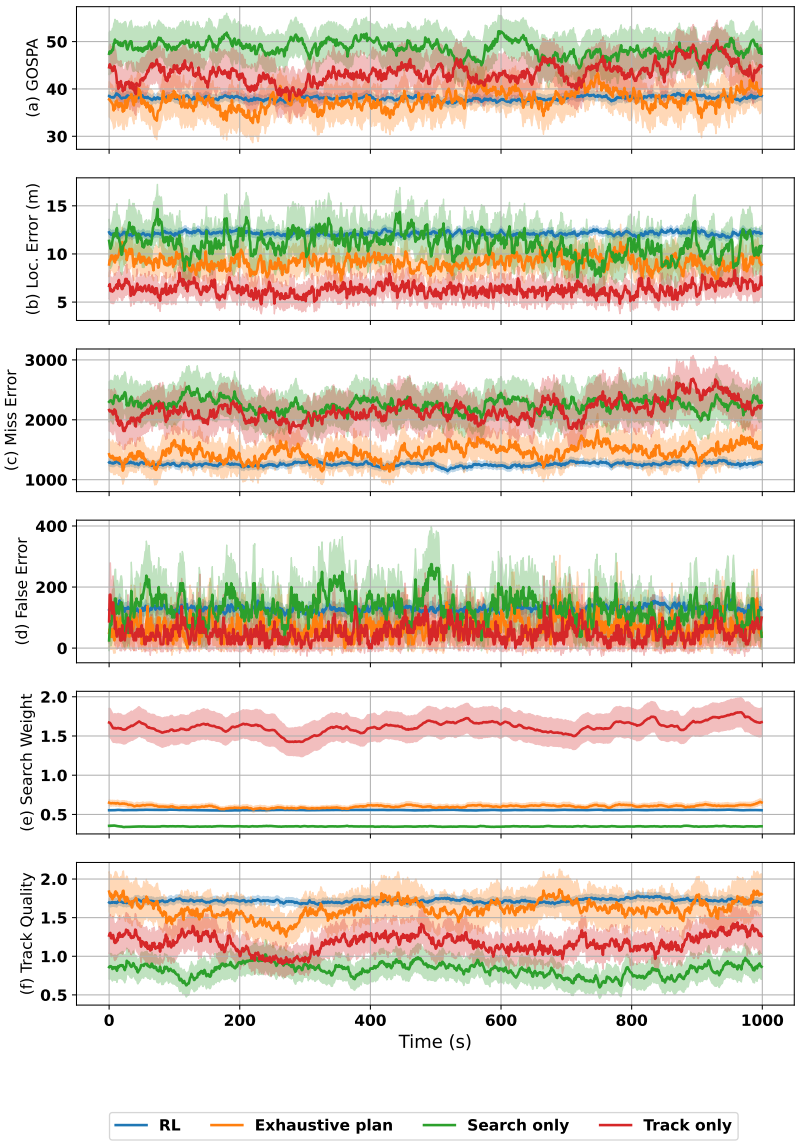


Figure 4.6: Search and Track performance metric for the beam optimization task. Three configurations of the RL agent (search+track, search-only, track-only) are compared to the exhaustive planning agent from [42] in terms of (a) GOSPA, (b) mean localization error, (c) Miss error, (d) False error, (e) Total undetected intensity weight, and (f) Total track quality.

Table 4.4: Search and track metrics for the beam optimization task.

Metric	RL (S+T)	RL (S)	RL (T)	Planning [42]
GOSPA	38.068	48.525	43.405	**37.671**
Loc. Error	12.144	11.036	**6.298**	9.163
False Error	129.223	130.625	**51.000**	68.475
Miss Error	**1258.829**	2250.512	2161.725	1453.612
Search weight	0.555	**0.347**	1.619	0.602
Track Quality	**1.719**	0.825	1.171	1.615

t-test was performed to compare the search and track performance of the RL and planning techniques. The null hypothesis of the test states that the mean GOSPA of the S+T RL agent equals the mean GOSPA of the planner. The resulting t-value is 3.640 with 19 degrees of freedom. The corresponding p-value is 1.74×10^{-3}, indicating that the small GOSPA difference between the two methods is statistically significant for a significance level $\alpha = 0.05$. Since the exhaustive planner is nearly optimal for the beam optimization scenario, this result suggests that the graph RL agent successfully learns a performant policy for the joint search and track task.

As expected, the track-only agent consistently maintains the lowest localization error on initiated tracks, followed by the exhaustive planning agent (Figure 4.6b). Figure 4.6c shows that the S+T RL agent sacrifices slightly higher per-track localization error to maintain more tracks than all other configurations (Figure 4.6c). The search-only agent has a similar localization error but fails to actively maintain tracks, producing a high miss error and a correspondingly high overall GOSPA error. Although track deletion is not explicitly optimized by either method, all configurations maintain low false track error and effectively remove false tracks (Figure 4.6d).

Figure 4.6e shows that the S+T and planning agent perform similarly on the

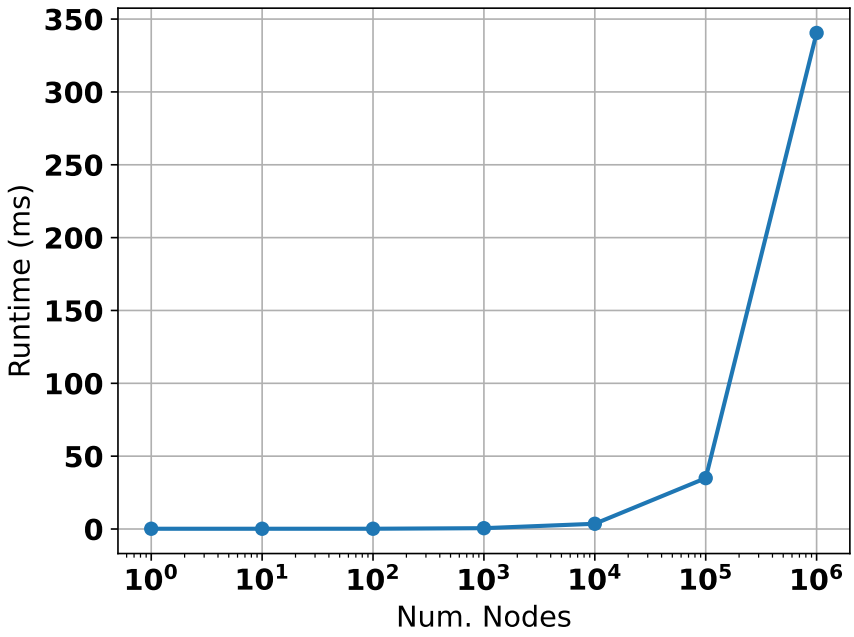


Figure 4.7: Scaling performance of the encoder network as a function of the input graph size (assuming 3 edges per node). Runtime complexity scales linearly with the size of the graph, and the encoder maintains sub-millisecond inference time for graphs with fewer than 10^4 nodes.

search task, reducing the search intensity weight to 0.555 and 0.602, respectively. The tracking performance of each agent is also similar, with the S+T RL agent obtaining a slightly higher overall track quality of 1.719 compared to 1.61 for the planning agent (Figure (4.6f)). The search-only agent maintains a low search intensity weight of 0.347 but fails to maintain tracks, producing the lowest track quality overall. The track-only agent performs poorly on the search task and fails to initiate tracks on many objects, resulting in a lower overall track quality than the S+T and planning agents. Table 4.4 summarizes these trends by reporting the mean value of each metric across all Monte Carlo episodes. The bold values in the table indicate the configuration with the highest mean value across all trials for each metric.

The final experiments in this section compare the scalability and runtime

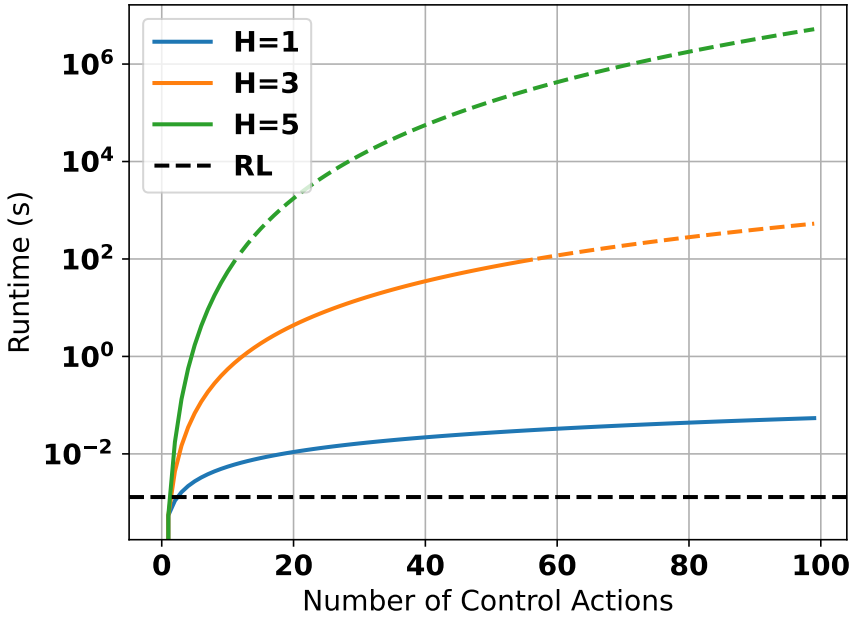


Figure 4.8: Exhaustive planner runtime analysis. The computational requirements of the exhaustive planner from [42] scale exponentially with the planning horizon (colored lines). The graph RL agent maintains a constant runtime of 1.2 ms per decision during inference (dashed black line).

computational requirements of the planning and RL agents. Figure 4.7 shows the inference time of the graph encoder as a function of the the input graph size on an RTX 4090 GPU. We assume that each node in the graph has three incoming edges on average, which is typical for the sparse graphs encountered in the beam optimization scenario. For small graphs with fewer than 1000 nodes, the GPU is memory-bound and the runtime is approximately constant. Beyond this threshold, the runtime complexity scales *linearly* with the graph size and the encoder maintains sub-millisecond inference time for graphs with up to 10 k nodes. For reference, the graphs used for training the RL agent contain approximately 100 nodes each.

Figure 4.8 shows the runtime of the exhaustive planner from [42] as a function of the planning horizon and number of feasible control inputs. All values in the

figure are averaged over 100 time steps in the beam optimization scenario on an AMD Ryzen Threadripper PRO 3955WXs CPU. To avoid excessive simulation times, the dashed colored lines represent extrapolated values whose runtime exceeds 100 s per decision. The approach from [42] scales *exponentially* with the horizon, quickly becoming infeasible even for modestly complex planning problems.

Since the graph RL agent uses dynamic programming to control its optimization horizon rather than explicit planning, its runtime does not scale with the horizon. The RL approach also plans in a continuous action space and its runtime does not depend on a pre-defined discretization of the control inputs. These properties make the graph RL approach drastically more efficient than the exhaustive planner for all possible configurations. Recent work has also shown that the runtime complexity of TD-MPC2 can be further reduced by with imitation learning techniques [85].

4.3.2 Mobile Surveillance Scenario

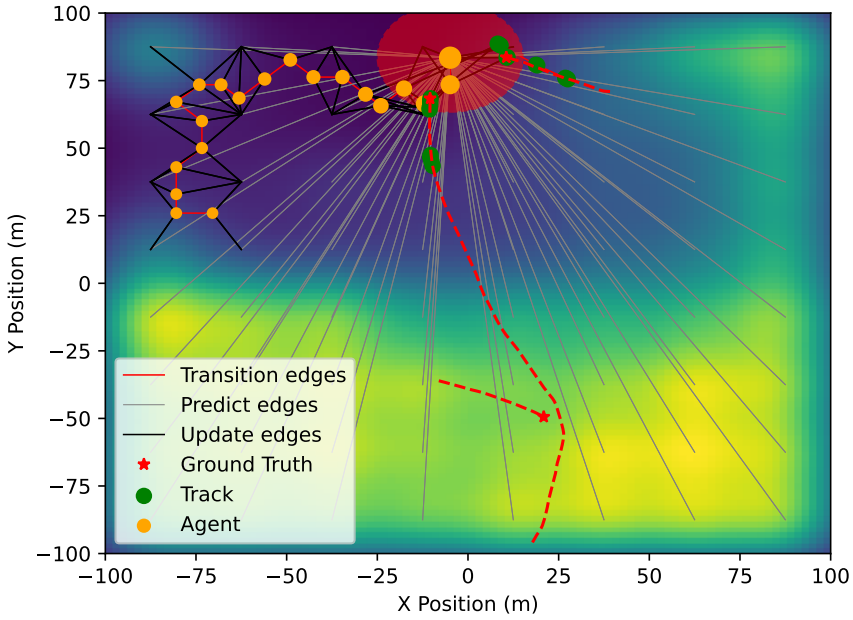


Figure 4.9: One realization of the mobile surveillance scenario. The sensing platform (yellow circles) maintains tracks (green circles) on two ground truth objects (red stars) and has yet to detect a third object in the bottom half of the surveillance region. For visual clarity, older states in the platform trajectory have smaller circle markers than more recent states. Gray lines represent prediction edges between agent nodes and search nodes.

The next scenario considers a mobile sensing platform that patrols a 200 m × 200 m area to localize objects of interest (Figure 4.9). At each decision step, the autonomous agent updates the platform speed and heading via the control input $u_t = [\Delta v_t, \Delta\psi_t]$, where Δv_t and $\Delta\psi_t$ are the change in speed and heading, respectively. The platform has a maximum speed of 10 m/s, can accelerate or decelerate at up to 3 m/s^2, and can turn at a maximum rate of $\pi/2$ rad/s.

The kinematic state of objects in the environment evolve under a coordinated turn (CT) model. The CT model assumes that objects move with constant speed while executing a turn with constant angular velocity [95]. In the 2-D case, the

CT state vector is defined as $x = [p_x, v_x, p_y, v_y, \omega]$, where (p_x, p_y) and (v_x, v_y) is the 2-D position and velocity, respectively, and ω is the turn rate. The transition density is Gaussian-distributed with $p(x_{t+1}|x_t) = \mathcal{N}(x_{t+1}; F_{CT}x_t, Q_{CT})$, where the state transition matrix is given by:

$$F_{CT} = \begin{bmatrix} 1 & \frac{\sin(\omega\Delta t)}{\omega} & 0 & -\frac{1-\cos(\omega\Delta t)}{\omega} & 0 \\ 0 & \cos(\omega\Delta t) & 0 & -\sin(\omega\Delta t) & 0 \\ 0 & \frac{1-\cos(\omega\Delta t)}{\omega} & 1 & \frac{\sin(\omega\Delta t)}{\omega} & 0 \\ 0 & \sin(\omega\Delta t) & 0 & \cos(\omega\Delta t) & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.17}$$

and the process noise covariance is a matrix of the form:

$$Q_{CT} = \begin{bmatrix} q_a UU^T & 0 \\ 0 & q_\omega \end{bmatrix}, \quad U = I_2 \otimes \begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix}, \tag{4.18}$$

where q_a and q_ω are the process noise intensities for the linear acceleration and turn rate, respectively. This scenario assumes $q_a = 3 \times 10^{-4}$ and $q_\omega = (0.01\pi/180)^2$.

Similar to the previous experiment, the birth model is a Gaussian mixture with means $\mu^{b,i} = \left[p_x^{b,i}, v_x^{b,i}, p_y^{b,i}, v_y^{b,i}, \omega^{b,i}\right]$, where $p_x^{b,i}$ and $p_y^{b,i}$ form a uniform 8×8 grid in position and $v_x^{b,i} = v_y^{b,i} = \omega^{b,i} = 0$. Each birth mixture component has an identical covariance matrix $P^{b,i} = \text{diag}\left([200/8, 1, 200/8, 1, \pi/180]\right)^2$ and weight $w^{b,i} = (1/100)/64 = 1.562 \times 10^{-4}$. Again, the survival probability is a uniform $p_s = 0.997$ within the surveillance region and zero outside. New objects appear in the simulation according to this birth model such that a new object is born every 100 s on average.

The mobile platform is equipped with a sensor that operates with a limited

range of $r_{max} = 20\,\text{m}$. The sensor's detection probability is a Gaussian function of the object's distance from the platform when the object lies within the sensor range and zero otherwise:

$$p_d(x^{(i)}) = \begin{cases} \exp\left[-\frac{1}{2}\left(\frac{r}{r_{max}}\right)^2\right] & r \leq r_{\max}, \\ 0 & r > r_{\max}, \end{cases} \tag{4.19}$$

where r is the Euclidean distance between the object and sensing platform. Unlike the beam optimization task in Section 4.3.1, the mobile sensor directly measures the Cartesian position of objects in the environment. The measurement likelihood function is a Gaussian density $p(z_t|x_t) = \mathcal{N}(z_t; Hx_t, R_t)$ with:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \quad R_t = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}, \tag{4.20}$$

where σ_x and σ_y are the measurement errors for the x and y position components, respectively.

Table 4.5 summarizes the relevant scenario parameters of the mobile surveillance task. Note that the RL and graph encoder parameters are identical to those in Section 4.3.1 aside from the discount factor, which is increased to 0.99. This increased discount factor reflects the fact that the mobile surveillance task requires longer-term behavioral optimization since the platform requires many successive decision steps to reach distant goal points. In general, the graph RL approach requires minimal tuning across different scenarios aside from the discount and the underlying simulator used to generate experiences.

Figure 4.10 compares the training reward of the graph RL agent to the reward of the MCTS planning agent from [42]. The MCTS agent plans over a horizon

Table 4.5: Mobile Surveillance Scenario Parameters

Parameter	Value	Unit
Scenario extents	200×200	m
Max platform speed	10	m/s
Max sensor range	20	m
Max linear acceleration	3	m/s^2
Max turn rate	$\pi/2$	rad/s
Max localization error	10	m
Δt	1	s
X pos. measurement error	0.1	m
Y pos. measurement error	0.1	m
Acceleration process noise	3×10^{-4}	—
Turn rate process noise	$(0.01\pi/180)^2$	—
Average birth rate	0.01	s^{-1}
Birth model	Uniform Poisson	—
Birth grid size	8×8	—
Survival probability	0.997	—
RL discount factor	0.99	—

of $H = 10$ time steps, discretizes the control space into 5 uniformly spaced heading bins and 4 speed bins, and performs 100 MCTS iterations per decision step. This MCTS configuration represents a reasonable trade-off between task performance and computational complexity and requires approximately 1.5 s per decision. Unlike the beam optimization task from Section 4.3.1, the RL agent ultimately achieves a larger reward than the MCTS planner after $2M$ training steps.

Figure 4.11 plots the search and track metrics for the MCTS planning agent and the RL configurations discussed in Section 4.3.1 (search-only, track-only, S+T). As before, the S+T RL and planning agents achieve the lowest GOSPA errors overall, achieving GOSPA values of 7.209 and 7.560, respectively (Figure 4.11a). The S+T RL agent tolerates slightly higher localization errors (Figure 4.11b) than the planning agent (1.743 and 1.119, respectively). However,

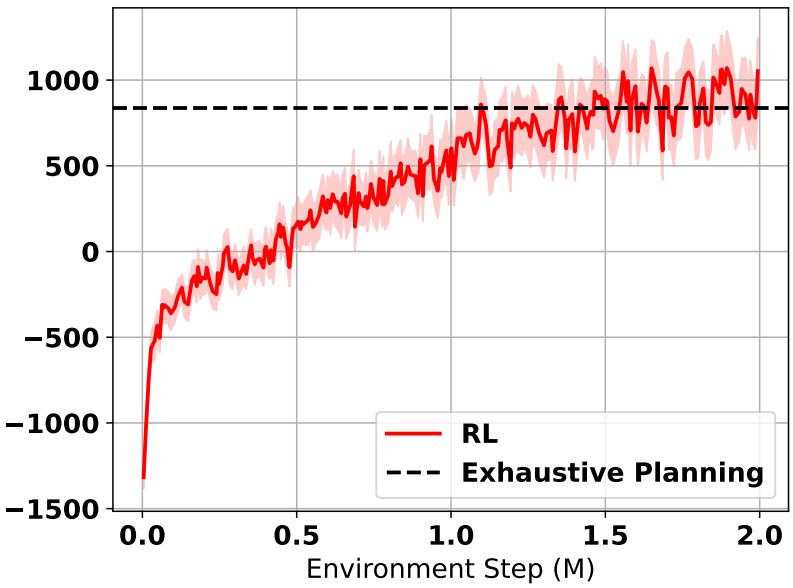


Figure 4.10: Learning curves for the mobile surveillance task over 20 random seeds. The black line gives the average reward of the MCTS planning agent from [42] over 100 Monte Carlo episodes. The RL agent ultimately achieves a higher reward than MCTS after training is complete.

the RL agent drops fewer tracks than all other configurations (Figure 4.11c). and achieves an average miss error of 49.273. As a result, the RL agent maintains a greater overall track quality than all other configurations on average (Figure 4.11f). Unlike in the beam optimization task, the planning agent consistently maintains a slightly lower undetected intensity weight than the S+T RL agent (Figure 4.11e). Aside from the search-only agent, all configurations effectively delete false tracks from the filter (Figure 4.11d) and maintain false errors below 10. Table 4.6 reports the average value of each metrics across all experiments.

To verify the statistical significance of these results, a one-tailed t-test was performed to compare the joint task performance of the RL and planning agents. As in Section 4.3.1, the null hypothesis of the test states that the GOSPA of

Table 4.6: Search and track metrics for the mobile surveillance task.

Metric	RL (S+T)	RL (S)	RL (T)	Planning [42]
GOSPA	**7.209**	10.422	9.105	7.560
Loc. Error	1.743	2.213	**0.175**	1.199
False Error	4.943	15.850	**0.864**	3.347
Miss Error	**49.273**	92.520	98.237	62.554
Search weight	0.862	**0.645**	1.363	0.802
Track Quality	**2.007**	0.983	0.973	1.756

the S+T RL agent equals the GOSPA of the MCTS planner. In this case, the alternate hypothesis states that the GOSPA of the RL agent *exceeds* that of the MCTS planner. With 20 independent S+T RL agents evaluated over 100 Monte Carlo episodes each, the resulting t-value is 9.734 with 19 degrees of freedom. The p-value is 4.051×10^{-9}, indicating that the GOSPA difference between the two methods is statistically significant for a significance level of $\alpha = 0.05$.

Figures 4.11e and 4.11f demonstrate more significant differences in the behavior of the S+T RL agent and the MCTS planner. The MCTS planner places greater emphasis on the search task and reduces the undetected intensity to an average value of 0.835 compared to 0.986 for the S+T agent (Figure 4.11e). Meanwhile, the S+T agent attains an average total track quality of 2.15 compared to 1.66 for the MCTS planner (Figure 4.11f). This indicates that the S+T agent maintains high quality tracks on more objects than the other configurations on average.

This experiment concludes by comparing the computational complexity of the MCTS planner to the graph RL agent. Figure 4.12 shows the average runtime of the MCTS planner as a function of the planning horizon and number of MCTS iterations. Unlike the exhaustive search planner, the runtime of MCTS is independent of the number of feasible control inputs and scales linearly with

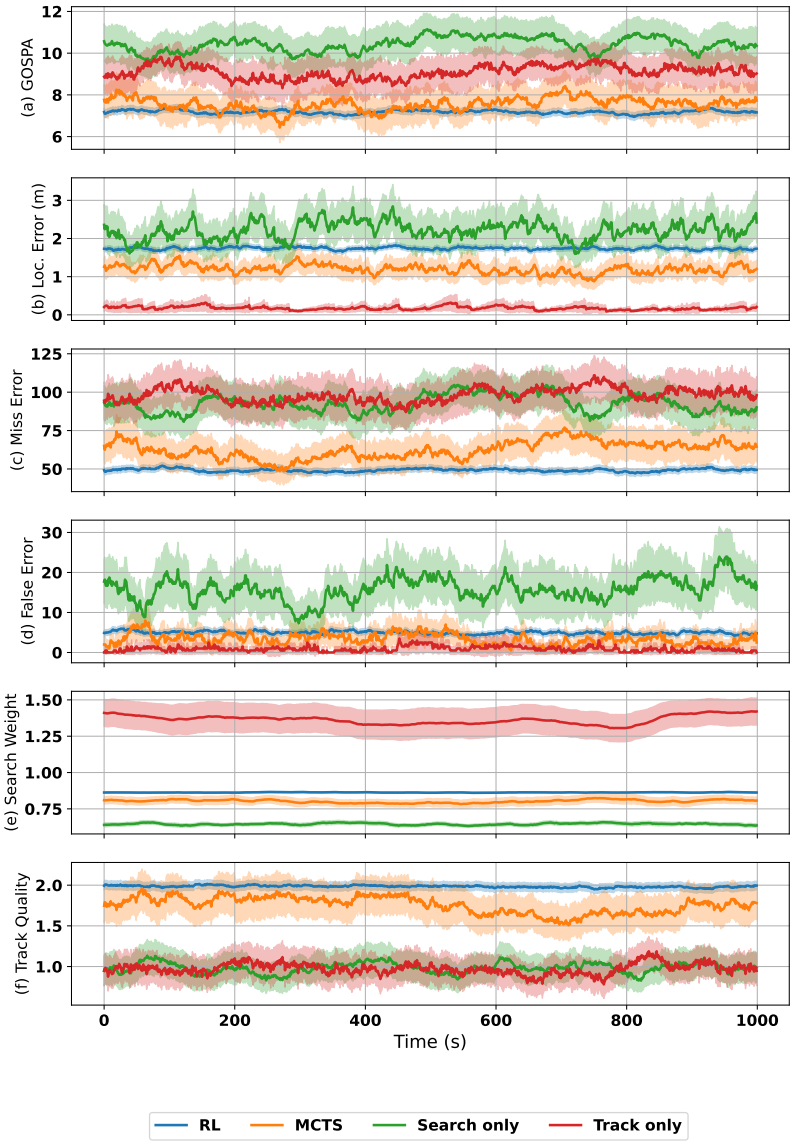


Figure 4.11: Search and Track performance metric for the mobile surveillance task. Three configurations of the RL agent (search+track, search-only, track-only) are compared to the MCTS planning agent from [42] in terms of (a) GOSPA, (b) mean localization error, (c) Miss error, (d) False error, (e) Total undetected intensity weight, and (f) Total track quality.

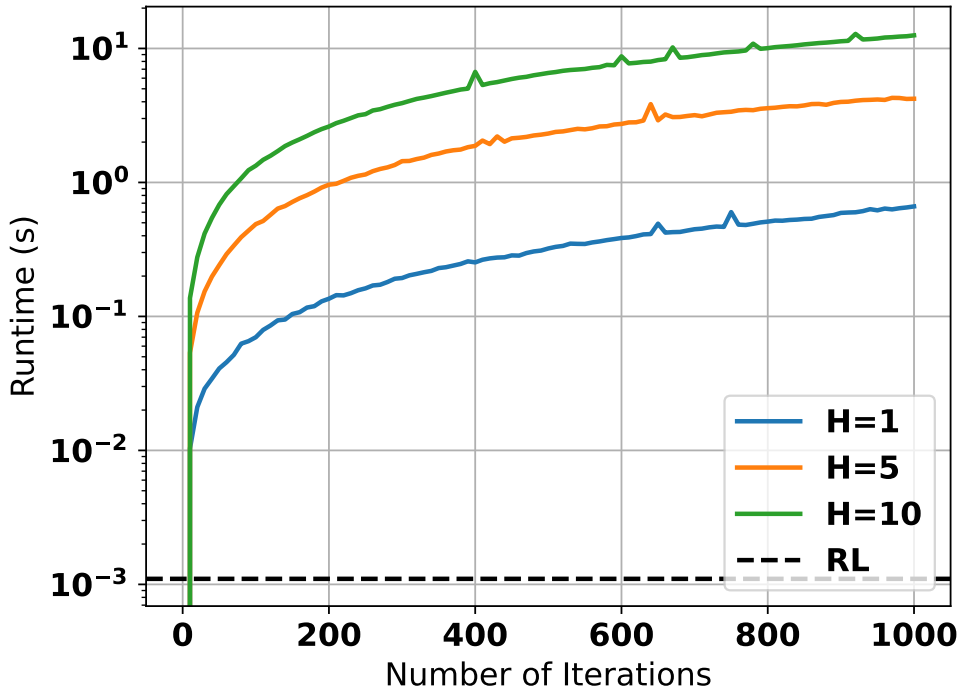


Figure 4.12: Planning runtime complexity vs planning horizon.

the planning horizon and number of iterations. However, the MCTS planner still scales poorly compared to the RL agent and requires > 1 s per decision for even modest planning horizons and iteration counts. Though the runtime complexity of MCTS can be improved by heavily optimizing the simulator and performing MCTS rollouts in parallel, it is unlikely that these optimizations would reduce the runtimes by several orders of magnitude to match the 1.2 ms latency of the RL agent.

4.4 Summary

This chapter has presented a novel reinforcement learning solution to the joint search and track control problem. First, a graphical representation of the general class of the multi-object tracking problem was developed. These graphs model

the complete history of sensor states and object tracks in a flexible structure that accommodates dynamic, time-varying sensor control scenarios. The graph structure was shown to be sensor agnostic and suitable for processing with graph neural networks. Next, a GNN architecture was developed to encode the scenario graphs into a compact latent representation for downstream learning tasks. Due to the inherent sparsity of the graph representation, the computational complexity of the GNN encoder scales linearly with the scenario history and size. Finally, a novel reward function was derived to unify the search and track objectives into a single optimization criterion.

The performance of the proposed graph RL agent was evaluated in two simulation studies. In the first experiment, the autonomous agents controlled a stationary sensor with an electronically steered beam. The second scenario considered a mobile platform tasked with patrolling a region using a range-limited sensor. In both scenarios, the graph RL agent matches or outperforms state-of-the-art planning techniques from the literature on the joint search and track task. At the same time, the proposed technique leverages world model planning to reduce inference-time computational requirements by several orders of magnitude compared to online planning methods. The experiments also showcase the graph RL architecture's ability to remain performant in highly disparate scenarios with minimal hyperparameter tuning. These results make the graph RL approach a promising option for resource-constrained sensor control regardless of the specific sensor or scenario.

Chapter 5

Conclusion

5.1 Summary

This dissertation developed several deep RL techniques for autonomous sensor control applications. Traditional control techniques often rely on myopic single-step heuristics, computationally intensive online planners, or machine learning methods that cannot adapt to complex, dynamic environments. This work aims to demonstrate that combining deep RL with relational reasoning and continuous control can overcome these limitations and provides a powerful general framework for autonomous sensor control. In particular, deep RL enables the cognitive agent to learn strategies that optimize long-term performance directly from experience. Relational reasoning makes it possible to model complex interactions between entities in the environment and to encode useful inductive biases of the problem directly into the learning representation. Finally, continuous control formulations provide maximum flexibility in the action selection process, allowing the agent to select control actions with as much precision as the problem requires. This hypothesis was evaluated in two distinct application spaces: radar-communications spectrum sharing and autonomous search and track. In both cases, the proposed relational RL techniques outperform state-of-the-art

baselines in terms of runtime complexity, final task performance, or both.

5.2 Novel Contributions

To address the shortcomings of the existing literature, this dissertation proposed the following contributions for the spectrum sharing task:

1. A POMDP formulation for the spectrum sharing task was developed to solve the problem as a *high-dimensional continuous control* task. The agent processes its frequency channel observations at the full resolution of the spectrum sensing process without downsampling. To fully utilize these degrees of freedom and maximize flexibility, the agent optimizes its waveform parameters from a continuous set.

2. A novel reward function was derived to balance radar performance and mutual interference avoidance across entire coherent processing intervals. This reward function is shown to smoothly trade bandwidth utilization, interference avoidance, and waveform coherence with minimal tuning across dynamic channel statistics.

3. A novel recurrent-attention network architecture was presented for radar-based spectrum access. The network is designed to exploit the periodic structure of pulsed radar data to efficiently process high-resolution spectrum data over extended time horizons.

4. The RL spectrum agent was evaluated using over-the-air data collected from a software-defined radio testbed. The agent is shown to perform favorably compared to the existing state-of-the-art in both the 2.4-2.5 GHz and 2.59-2.69 GHz bands.

For the autonomous search and track task, this work makes the following contributions:

1. A novel graphical representation was developed for the general class of search and track problems. The graph representation models spatiotemporal relationships between sensing platforms, tracked objects, and search regions of interest. The representation is sensor-agnostic and accommodates scenarios with a variable number of entities (agents, tracks, and search regions) during runtime.

2. A graph neural network (GNN) architecture was presented for processing time-varying scenario graphs. Since the graph representation is sparse, the GNN processes graphs with approximately *linear* time complexity in the number of entities in the multi-object tracking filter.

3. A novel reward function was derived to adaptively balance the search and track tasks. The reward function requires minimal tuning across sensor types and can be computed directly from the state estimate of a multi-object tracking filter.

4. The search and track task was formulated as a planning problem in the continuous control domain. Unlike discrete planning approaches, the proposed planner can operate in large control spaces without incurring additional computational cost.

5. Recent advances in model-based RL were leveraged to perform planning in a learned *world model* rather than the true environment simulator. The planning procedure is also augmented with a learned value model that allows the graph agent to optimize its behavior over long time horizons without

explicit simulation. As a result, the overall decision-making pipeline is orders of magnitude faster than state-of-the-art planning techniques, making it highly suitable for real-time operation.

6. The TOMB/P multi-object tracking filter was used to represent the environment state for the autonomous search and track agent. This design choice further reduces the latency for decision-making since the TOMB/P filter performs data association with linear time complexity in the number of tracks and measurements. It also significantly reduces the time required to train the agent using commercial hardware.

5.3 Future Work

This dissertation provides a foundation for future research into applications of deep reinforcement learning to autonomous sensor control. Several promising avenues for future work are outlined below for the two application spaces considered in this work.

5.3.1 Radar-Communications Spectrum Sharing

Future work could expand upon several aspects of the cognitive radar spectrum sharing formulation from Chapter 3. For instance, the reward function in Equation (3.4) effectively balances the spectrum sharing and radar performance objectives but requires a degree of manual tuning and does not directly optimize task-level performance metrics of interest. Future work could therefore explore more principled reward structures that more directly express the trade-offs between each objective. As a starting point, the distortion reward in Equation (3.3) is expressed in terms of waveform adaptation metrics that are only a proxy for

coherent radar performance. The distortion reward could instead derive directly from characteristics of the range-Doppler PSF such as the peak sidelobe level (PSL) or integrated sidelobe level (ISL) [128]. Alternative rewards based on image processing metrics such as the structural similarity index measure (SSIM) [129] could also be used to compare the ideal fully coherent PSF to the pulse-agile response.

Similarly, the spectrum reward from Equation (3.1) characterizes mutual interference in terms of the collision bandwidth. Future reward functions could incorporate direct feedback from the radar detection/tracking process or from communications quality-of-service metrics. Given a set of task-level quality measures for each objective, the RL agent could also select additional radar parameters (e.g., the pulse width and pulse repetition frequency) in addition to the waveform's frequency content to optimize the overall performance of the system.

The architecture of the cognitive agent itself could also be improved in future work. The recurrent-attention architecture presented in Chapter 3 was designed for computational efficiency and does not include several recent advances in transformer network design (e.g., skip connections and layer normalization). Novel network architectures could explore the use of set transformers and pooling by multi-head attention (PMA) techniques to combine attention processing and pooling into a single module [121]. Scalable transformer architectures could also be investigated to eliminate the recurrent component entirely [130]. Future work must carefully consider the real-time performance of novel network architectures on software-defined platforms. Since pulse-agile radar systems operate on microsecond-level timescales, an architecture that is performant on offline training data may react too slowly in realistic environments.

5.3.2 Graph Learning for Sensor Control

The graph learning approach presented in Chapter 4 can also be extended in several ways. The experiments in Chapter 4 use a single autonomous sensing platform to perform the joint search and track task. However, the graph formulation can naturally accommodate a time-varying number of sensors as well.Figure 5.1 illustrates one possible graph structure for multi-agent operation. In this formulation, the graph stores the trajectories of N_s sensors that operate independently. If data fusion is performed at a central processor, each sensor could share an identical environment graph at each time step. For decentralized communication-limited operation, each sensor could maintain its own local graph that is periodically updated when the agent communicates with other agents. Agent-to-agent interactions could be modeled as an additional edge type in the graph (blue edges in Figure 5.1). Future work could investigate the scalability of the multi-agent graph RL approach as a function of the number of active sensors and the degree of communication between agents.

Multi-task operation is a major aspect of the TD-MPC2 algorithm that was not explored in this work. Rather than training a separate agent for each scenario or sensing modality, future work could exploit the structural similarity between sensor control problems to train a single generalist agent that seamlessly operates across multiple tasks. To this end, future work could investigate graphical *foundation models* for sensor control that are pre-trained on a diverse corpus of scenarios and fine-tuned for specific applications as needed. Augmenting the existing agent to support multi-task operation would likely require minimal changes to the network architecture beyond scaling the size of the world model and conditioning all model components on the current task.

Finally, future work should deploy the graph RL agent on a real autonomous

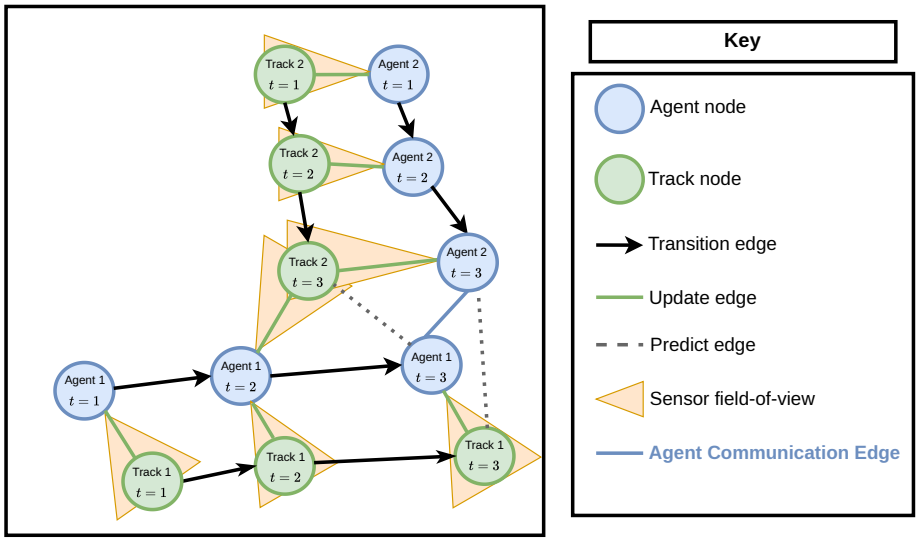


Figure 5.1: Proposed multi-sensor extension to the graphical search and track approach. Two autonomous platforms (agents 1 and 2) maintain a shared graph state. The multi-agent formulation extends the single-agent graph with separate trajectory nodes for each sensor and inter-agent communication edges (blue). Search nodes are omitted for clarity.

platform to evaluate its performance under real-world constraints. Since the graph RL agent operates on millisecond-level inference timescales, it should be straightforward to incorporate into a robotic platform control loop (where 50 Hz operation is standard). Since the graph agent operates on highly processed tracker outputs, it should be robust to sensor artifacts and other deviations from the simulation model. Since the search and track reward function defined by Equations (4.1)-(4.4) does not require ground truth information, the deployed agent could be trained directly on real situational data with minimal direct supervision.

References

[1] A. B. Charlish, "Autonomous agents for multi-function radar resource management," Ph.D. dissertation, UCL (University College London), 2011.

[2] A. Charlish and F. Katsilieris, "Array radar resource management," *Novel radar techniques and applications: real aperture array radar, imaging radar, and passive and multistatic radar*, vol. 1, pp. 135–171, 2017.

[3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005, ISBN: 0-262-20162-3.

[4] H. Griffiths, L. Cohen, S. Watts, *et al.*, "Radar Spectrum Engineering and Management: Technical and Regulatory Issues," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 85–102, Jan. 2015, ISSN: 1558-2256. DOI: 10.1109/JPROC.2014.2365517.

[5] S. Blunt, *Radar and Communication Spectrum Sharing*. IET Digital Library, Oct. 2018, ISBN: 978-1-78561-358-6. DOI: 10.1049/SBRA515E.

[6] A. F. Martone, K. D. Sherbondy, J. A. Kovarskiy, *et al.*, "Closing the Loop on Cognitive Radar for Spectrum Sharing," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 36, no. 9, pp. 44–55, Sep. 2021, ISSN: 0885-8985, 1557-959X. DOI: 10.1109/MAES.2021.3072698.

[7] P. Stinco, M. S. Greco, and F. Gini, "Spectrum sensing and sharing for cognitive radars," *IET Radar, Sonar & Navigation*, vol. 10, no. 3, pp. 595–602, 2016.

[8] A. W. Clegg, S. A. Seguin, R. J. Marks, and C. Baylis, "Radar sharing in the U.S. 3 ghz band," in *2022 IEEE Radar Conference (RadarConf22)*, 2022, pp. 1–5. DOI: 10.1109/RadarConf2248738.2022.9764348.

[9] M. Ghosh, "The fcc's auction authority, the 3.1–3.45 ghz conundrum, and ntia's spectrum strategy," *IEEE Wireless Communications*, vol. 30, no. 2, pp. 6–7, 2023.

[10] S. Bhattarai, J.-M. J. Park, B. Gao, K. Bian, and W. Lehr, "An overview of dynamic spectrum sharing: Ongoing initiatives, challenges, and a roadmap for future research," *IEEE Transactions on Cognitive Communications and Networking*, vol. 2, no. 2, pp. 110–128, 2016.

[11] S. Haykin, "Cognitive radar: A way of the future," *IEEE Signal Processing Magazine*, vol. 23, no. 1, pp. 30–40, Jan. 2006, ISSN: 1558-0792. DOI: 10.1109/MSP.2006.1593335.

[12] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016. DOI: 10.1109/TIV.2016.2578706.

[13] K. Rado, M. Baglioni, and A. Jamshidnejad, "Enabling robots to autonomously search dynamic cluttered post-disaster environments," *Sci Rep*, vol. 15, no. 1, p. 34 778, Oct. 2025, ISSN: 2045-2322. DOI: 10.1038/s41598-025-18573-y.

[14] H. Van Nguyen, B.-N. Vo, B.-T. Vo, H. Rezatofighi, and D. C. Ranasinghe, "Multi-objective multi-agent planning for discovering and tracking multiple mobile objects," *IEEE Transactions on Signal Processing*, vol. 72, pp. 3669–3685, 2024.

[15] R. P. S. Mahler, *Statistical Multisource-Multitarget Information Fusion* (Artech House Information Warfare Library). Boston, Mass. London: Artech House, 2007, ISBN: 978-1-59693-092-6.

[16] R. P. S. Mahler, *Advances in Statistical Multisource-Multitarget Information Fusion* (Artech House Electronic Warfare Library). Boston: Artech House, 2014, ISBN: 978-1-60807-798-4.

[17] B.-N. Vo and W.-K. Ma, "The Gaussian mixture probability hypothesis density filter," *IEEE Transactions on signal processing*, vol. 54, no. 11, pp. 4091–4104, 2006.

[18] Á. F. García-Fernández, J. L. Williams, K. Granström, and L. Svensson, "Poisson multi-Bernoulli mixture filter: Direct derivation and implementation," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 54, no. 4, pp. 1883–1901, Aug. 2018, ISSN: 0018-9251, 1557-9603, 2371-9877. DOI: 10.1109/TAES.2018.2805153. arXiv: 1703.04264 [cs, stat].

[19] J. L. Williams, "Marginal multi-bernoulli filters: RFS derivation of MHT, JIPDA, and association-based member," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 51, no. 3, pp. 1664–1687, Jul. 2015, ISSN: 1557-9603. DOI: 10.1109/TAES.2015.130550.

[20] F. Meyer, T. Kropfreiter, J. L. Williams, *et al.*, "Message Passing Algorithms for Scalable Multitarget Tracking," *Proceedings of the IEEE*, vol. 106, no. 2, pp. 221–259, Feb. 2018, ISSN: 1558-2256. DOI: 10.1109/JPROC.2018.2789427.

[21] S. S. Blackman, "Multiple-target tracking with radar applications," *Dedham*, 1986.

[22] Y. Bar-Shalom, F. Daum, and J. Huang, "The probabilistic data association filter," *IEEE Control Systems Magazine*, vol. 29, no. 6, pp. 82–100, Dec. 2009, ISSN: 1941-000X. DOI: 10.1109/MCS.2009.934469.

[23] A. F. Martone, K. I. Ranney, K. Sherbondy, K. A. Gallagher, and S. D. Blunt, "Spectrum Allocation for Noncooperative Radar Coexistence," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 1, pp. 90–105, Feb. 2018, ISSN: 1557-9603. DOI: 10.1109/TAES.2017.2735659.

[24] B. H. Kirk, R. M. Narayanan, K. A. Gallagher, A. F. Martone, and K. D. Sherbondy, "Avoidance of time-varying radio frequency interference with software-defined cognitive radar," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 55, no. 3, pp. 1090–1107, 2018.

[25] A. Martone and K. Ranney, "Fast technique for wideband spectrum sensing," in *2014 IEEE Antennas and Propagation Society International Symposium (APSURSI)*, 2014, pp. 1206–1207. DOI: 10.1109/APS.2014.6904930.

[26] J. W. Owen, C. A. Mohr, B. H. Kirk, S. D. Blunt, A. F. Martone, and K. D. Sherbondy, "Demonstration of real-time cognitive radar using spectrally-

notched random FM waveforms," in *2020 IEEE International Radar Conference (RADAR)*, IEEE, 2020, pp. 123–128.

[27] N. C. Luong, D. T. Hoang, S. Gong, *et al.*, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019. DOI: 10.1109/COMST.2019.2916583.

[28] Z. Du, Y. Deng, W. Guo, A. Nallanathan, and Q. Wu, "Green deep reinforcement learning for radio resource management: Architecture, algorithm compression, and challenges," *IEEE Vehicular Technology Magazine*, vol. 16, no. 1, pp. 29–39, 2021. DOI: 10.1109/MVT.2020.3015184.

[29] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, "Deep reinforcement learning for dynamic multichannel access in wireless networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 2, pp. 257–265, 2018. DOI: 10.1109/TCCN.2018.2809722.

[30] Y. Xu, J. Yu, and R. M. Buehrer, "The application of deep reinforcement learning to distributed spectrum access in dynamic heterogeneous environments with partial observations," *IEEE Transactions on Wireless Communications*, vol. 19, no. 7, pp. 4494–4506, 2020. DOI: 10.1109/TWC.2020.2984227.

[31] O. Naparstek and K. Cohen, "Deep multi-user reinforcement learning for distributed dynamic spectrum access," *IEEE Transactions on Wireless Communications*, vol. 18, no. 1, pp. 310–323, 2019. DOI: 10.1109/TWC.2018.2879433.

[32] X. Tan, L. Zhou, H. Wang, *et al.*, "Cooperative multi-agent reinforcement-learning-based distributed dynamic spectrum access in cognitive radio networks," *IEEE Internet of Things Journal*, vol. 9, no. 19, pp. 19 477–19 488, 2022. DOI: 10.1109/JIOT.2022.3168296.

[33] Y. Lu, H. Lu, L. Cao, F. Wu, and D. Zhu, "Learning deterministic policy with target for power control in wireless networks," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–7. DOI: 10.1109/GLOCOM.2018.8648056.

[34] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, *Continuous control with deep reinforcement learning*, Jul. 2019. DOI: 10.48550/arXiv.1509.02971. arXiv: 1509.02971 [cs].

[35] F. Zishen, X. Xianzhong, S. Zhaoyuan, and C. Xiping, "Proximal policy optimization based continuous intelligent power control in cognitive radio network," in *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, 2020, pp. 820–824. DOI: 10.1109/ICCC51575.2020.9345062.

[36] E. Selvi, R. M. Buehrer, A. Martone, and K. Sherbondy, "On the use of Markov Decision Processes in cognitive radar: An application to target tracking," in *2018 IEEE Radar Conference (RadarConf18)*, 2018, pp. 0537–0542. DOI: 10.1109/RADAR.2018.8378616.

[37] E. Selvi, R. M. Buehrer, A. Martone, and K. Sherbondy, "Reinforcement Learning for Adaptable Bandwidth Tracking Radars," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 5, pp. 3904–3921, Oct. 2020, ISSN: 1557-9603. DOI: 10.1109/TAES.2020.2987443.

[38] M. Kozy, J. Yu, R. M. Buehrer, A. Martone, and K. Sherbondy, "Applying deep-q networks to target tracking to improve cognitive radar," in *2019 IEEE Radar Conference (RadarConf)*, 2019, pp. 1–6. DOI: 10.1109/RADAR.2019.8835780.

[39] C. E. Thornton, M. A. Kozy, R. M. Buehrer, A. F. Martone, and K. D. Sherbondy, "Deep Reinforcement Learning Control for Radar Detection and Tracking in Congested Spectral Environments," *IEEE Trans. Cogn. Commun. Netw.*, vol. 6, no. 4, pp. 1335–1349, Dec. 2020, ISSN: 2332-7731, 2372-2045. DOI: 10.1109/TCCN.2020.3019605. arXiv: 2006.13173 [eess].

[40] C. E. Thornton, R. M. Buehrer, and A. F. Martone, "Constrained online learning to mitigate distortion effects in pulse-agile cognitive radar," in *2021 IEEE Radar Conference (RadarConf21)*, IEEE, 2021, pp. 1–6.

[41] C. Thornton and R. Buehrer, "When is cognitive radar beneficial? Insights from dynamic spectrum access," in *2023 IEEE Radar Conference (RadarConf23)*, 2023, pp. 1–6. DOI: 10.1109/RadarConf2351548.2023.10149461.

[42] P. Boström-Rost, D. Axehill, and G. Hendeby, "Sensor Management for Search and Track Using the Poisson Multi-Bernoulli Mixture Filter," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 57, no. 5, pp. 2771–2783, Oct. 2021, ISSN: 1557-9603. DOI: 10.1109/TAES.2021.3061802.

[43] C. B. Browne, E. Powley, D. Whitehouse, *et al.*, "A Survey of Monte Carlo Tree Search Methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012, ISSN: 1943-068X, 1943-0698. DOI: 10.1109/TCIAIG.2012.2186810.

[44] H. Van Nguyen, B.-N. Vo, B.-T. Vo, H. Rezatofighi, and D. C. Ranasinghe, *Multi-Objective Multi-Agent Planning for Discovering and Tracking Multiple Mobile Objects*, https://arxiv.org/abs/2203.04551v3, Mar. 2022.

[45] J. Olofsson, G. Hendeby, T. R. Lauknes, and T. A. Johansen, "Multi-agent informed path planning using the probability hypothesis density," *Autonomous Robots*, vol. 44, no. 6, pp. 913–925, Jul. 2020, ISSN: 1573-7527. DOI: 10.1007/s10514-020-09904-1.

[46] J. A. Placed, J. Strader, H. Carrillo, *et al.*, "A survey on active simultaneous localization and mapping: State of the art and new frontiers," *Trans. Rob.*, vol. 39, no. 3, pp. 1686–1705, Jun. 2023, ISSN: 1552-3098. DOI: 10.1109/TRO.2023.3248510.

[47] S. Thrun and M. Montemerlo, "The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, May 2006, ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364906065387.

[48] V. Indelman, L. Carlone, and F. Dellaert, "Planning in the continuous domain: A generalized belief space approach for autonomous navigation in unknown environments," *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 849–882, 2015.

[49] F. Chen, J. D. Martin, Y. Huang, J. Wang, and B. Englot, "Autonomous exploration under uncertainty via deep reinforcement learning on graphs," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 6140–6147.

[50] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, 6000–6010, ISBN: 9781510860964.

[51] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer normalization*, 2016. arXiv: 1607.06450 [stat.ML].

[52] R. Xiong, Y. Yang, D. He, *et al.*, "On layer normalization in the transformer architecture," in *Proceedings of the 37th International Conference on Machine Learning*, ser. ICML'20, vol. 119, JMLR.org, Jul. 2020, pp. 10 524–10 533.

[53] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[54] T. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.

[55] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy.

[56] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Trans. Neur. Netw.*, vol. 5, no. 2, 157–166, Mar. 1994, ISSN: 1045-9227. DOI: 10.1109/72.279181. [Online]. Available: https://doi.org/10.1109/72.279181.

[57] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[58] K. Cho, B. van Merriënboer, C. Gulcehre, *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, A. Moschitti, B. Pang, and W.

Daelemans, Eds., Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.

[59] P. Battaglia, J. B. C. Hamrick, V. Bapst, *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv*, 2018.

[60] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*, May 2021. DOI: 10.48550/arXiv.2104.13478. arXiv: 2104.13478 [cs].

[61] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl.

[62] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, 1025–1035, ISBN: 9781510860964.

[63] K. Oono and T. Suzuki, "Graph neural networks exponentially lose expressive power for node classification," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=S1ldO2EFPr.

[64] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" In *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ryGs6iA5Km.

[65] B. Weisfeiler and A. A. Lehman, "A Reduction of a Graph to a Canonical Form and an Algebra Arising During This Reduction," *Nauchno-Technicheskaya Informatsia*, vol. Ser. 2, no. N9, pp. 12–16, 1968.

[66] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, Lille, France: JMLR.org, 2015, pp. 448–456.

[67] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ.

[68] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" In *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=F72ximsx7C1.

[69] X. Bresson and T. Laurent, *Residual gated graph convnets*, 2018. arXiv: 1711.07553 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1711.07553.

[70] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *J. Mach. Learn. Res.*, vol. 24, no. 1, Jan. 2023, ISSN: 1532-4435.

[71] R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement Learning: An Introduction*. MIT press Cambridge, 1998, vol. 1.

[72] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. USA: John Wiley & Sons, Inc., 1994, ISBN: 0-471-61977-9.

[73] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, May 1998, ISSN: 00043702. DOI: 10.1016/S0004-3702(98)00023-X.

[74] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach Learn*, vol. 3, no. 1, pp. 9–44, Aug. 1988, ISSN: 0885-6125, 1573-0565. DOI: 10.1007/BF00115009.

[75] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal Policy Optimization Algorithms*, Aug. 2017. DOI: 10.48550/arXiv.1707.06347. arXiv: 1707.06347 [cs].

[76] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 229–256, May 1992, ISSN: 0885-6125. DOI: 10.1007/BF00992696.

[77] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang, "The 37 implementation details of proximal policy optimization," *The ICLR Blog Track*, 2022.

[78] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[79] C. Yu, A. Velu, E. Vinitsky, *et al.*, "The surprising effectiveness of PPO in cooperative multi-agent games," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., 2022, pp. 24 611–24 624.

[80] P. Wang, L. Li, Z. Shao, *et al.*, "Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, L.-W. Ku, A. Martins, and V. Srikumar, Eds., Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 9426–9439. DOI: 10.18653/v1/2024.acl-long.510.

[81] N. Hansen, H. Su, and X. Wang, "TD-MPC2: Scalable, robust world models for continuous control," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=0xh5CstDJU.

[82] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018.

[83] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse control tasks through world models," *Nature*, vol. 640, no. 8059, pp. 647–653, Apr. 2025, ISSN: 1476-4687. DOI: 10.1038/s41586-025-08744-2.

[84] N. Hansen, X. Wang, and H. Su, "Temporal Difference Learning for Model Predictive Control," arXiv, Jul. 2022. DOI: 10.48550/arXiv.2203.04955. arXiv: 2203.04955 [cs].

[85] Y. Wang, H. Guo, S. Wang, L. Qian, and X. Lan, "Bootstrapped model predictive control," in *The Thirteenth International Conference on Learning Representations*, 2025.

[86] S. Fujimoto, P. D'Oro, A. Zhang, Y. Tian, and M. Rabbat, "Towards General-Purpose Model-Free Reinforcement Learning," in *The Thirteenth International Conference on Learning Representations*, Oct. 2024.

[87] D. Misra, *Mish: A self regularized non-monotonic activation function*, 2020. arXiv: `1908.08681 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/1908.08681`.

[88] J. Farebrother, J. Orbay, Q. Vuong, *et al.*, "Stop Regressing: Training Value Functions via Classification for Scalable Deep RL," in *Proceedings of the 41st International Conference on Machine Learning*, PMLR, Jul. 2024, pp. 13 049–13 071.

[89] X. Chen, C. Wang, Z. Zhou, and K. W. Ross, "Randomized ensembled double q-learning: Learning fast without a model," in *International Conference on Learning Representations*, 2021. [Online]. Available: `https://openreview.net/forum?id=AY8zfZm0tDd`.

[90] T. Hiraoka, T. Imagawa, T. Hashimoto, T. Onishi, and Y. Tsuruoka, "Dropout q-functions for doubly efficient reinforcement learning," in *International Conference on Learning Representations*, 2022.

[91] S. Lavoie, C. Tsirigotis, M. Schwarzer, *et al.*, "Simplicial embeddings in self-supervised learning and downstream classification," in *International Conference on Learning Representations*, 2023. [Online]. Available: `https://openreview.net/forum?id=RWtGreRpovS`.

[92] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, Sydney, NSW, Australia: JMLR.org, 2017, 449–458.

[93] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.

[94] G. Williams, A. Aldrich, and E. Theodorou, "Model Predictive Path Integral Control using Covariance Variable Importance Sampling," *arXiv e-prints*, arXiv:1509.01149, arXiv:1509.01149, Sep. 2015. DOI: 10.48550/arXiv.1509.01149. arXiv: 1509.01149 [cs.SY].

[95] Y. Bar-Shalom, P. Willett, and X. Tian, *Tracking and Data Fusion: A Handbook of Algorithms*. YBS Publishing, 2011, ISBN: 978-0-9648312-7-8.

[96] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*. USA: Prentice-Hall, Inc., 1993, ISBN: 0-13-345711-7.

[97] D. Simon, *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. John Wiley & Sons, 2006.

[98] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, Mar. 1960, ISSN: 0021-9223. DOI: 10.1115/1.3662552. eprint: https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/82/1/35/5518977/35_1.pdf. [Online]. Available: https://doi.org/10.1115/1.3662552.

[99] 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)*, 2000, pp. 153–158. DOI: 10.1109/ASSPCC.2000.882463.

[100] S. Julier and J. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004. DOI: 10.1109/JPROC.2003.823141.

[101] R. Van Der Merwe, *Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models*. Oregon Health & Science University, 2004.

[102] W. H. Press, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge university press, 2007.

[103] S. Julier, "The scaled unscented transformation," in *Proceedings of the 2002 American Control Conference*, vol. 6, May 2002, 4555–4559 vol.6. DOI: 10.1109/ACC.2002.1025369.

[104] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002. DOI: 10.1109/78.978374.

[105] B.-T. Vo and B.-N. Vo, "Labeled Random Finite Sets and Multi-Object Conjugate Priors," *IEEE Transactions on Signal Processing*, vol. 61, no. 13, pp. 3460–3475, Jul. 2013, ISSN: 1941-0476. DOI: 10.1109/TSP.2013.2259822.

[106] T. Kropfreiter, F. Meyer, and F. Hlawatsch, "Sequential Monte Carlo implementation of the track-oriented marginal multi-Bernoulli/poisson filter," in *2016 19th International Conference on Information Fusion (FUSION)*, Jul. 2016, pp. 972–979.

[107] M. A. Richards, *Fundamentals of Radar Signal Processing*, 2nd Edition. McGraw-Hill Education, 2014, ISBN: 978-0-07-179832-7.

[108] T. Higgins, K. Gerlach, A. K. Shackelford, and S. D. Blunt, "Aspects of Non-Identical Multiple Pulse Compression," in *2011 IEEE RadarCon (RADAR)*, Kansas City, MO, USA: IEEE, May 2011, pp. 895–900, ISBN: 978-1-4244-8901-5. DOI: 10.1109/RADAR.2011.5960666.

[109] S. A. Flandermeyer, R. G. Mattingly, and J. G. Metcalf, "Deep Reinforcement Learning for Cognitive Radar Spectrum Sharing: A Continuous Control Approach," *IEEE Transactions on Radar Systems*, vol. 2, pp. 125–137, 2024, ISSN: 2832-7357. DOI: 10.1109/TRS.2024.3353112.

[110] R. G. Mattingly, A. F. Martone, and J. G. Metcalf, "Techniques for Mitigating the Impact of Intra-CPI Waveform Agility," *IEEE Transactions on Radar Systems*, vol. 2, pp. 24–40, 2024, ISSN: 2832-7357. DOI: 10.1109/TRS.2023.3343192.

[111] M. Andrychowicz, A. Raichuk, P. Stańczyk, *et al.*, "What matters for on-policy deep actor-critic methods? a large-scale study," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=nIAxjsniDzg.

[112] S. D. Blunt, J. K. Jakabosky, C. A. Mohr, *et al.*, "Principles and Applications of Random FM Radar Waveform Design," *IEEE Aerospace and*

Electronic Systems Magazine, vol. 35, no. 10, pp. 20–28, Oct. 2020, ISSN: 1557-959X. DOI: 10.1109/MAES.2019.2953763.

[113] T. Higgins, S. D. Blunt, and A. K. Shackelford, "Time-Range Adaptive Processing for pulse agile radar," in *2010 International Waveform Diversity and Design Conference*, Aug. 2010, pp. 115–120. DOI: 10.1109/WDD.2010.5592394.

[114] B. Kirk, A. Martone, K. Sherbondy, and R. Narayanan, "Mitigation of target distortion in pulse-agile sensors via Richardson–Lucy deconvolution," *Electronics Letters*, vol. 55, no. 23, pp. 1249–1252, 2019. DOI: 10.1049/el.2019.2650. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/el.2019.2650.

[115] C. Sahin, J. G. Metcalf, and S. D. Blunt, "Filter design to address range sidelobe modulation in transmit-encoded radar-embedded communications," in *2017 IEEE Radar Conference (RadarConf)*, Seattle, WA, USA: IEEE, May 2017, pp. 1509–1514, ISBN: 978-1-4673-8823-8. DOI: 10.1109/RADAR.2017.7944446.

[116] C. E. Thornton, R. M. Buehrer, and A. F. Martone, "Efficient Online Learning for Cognitive Radar-Cellular Coexistence via Contextual Thompson Sampling," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, Dec. 2020, pp. 1–6. DOI: 10.1109/GLOBECOM42002.2020.9322256.

[117] S. A. Flandermeyer and J. G. Metcalf, "Deep reinforcement learning on graphs for autonomous search-and-track," in *2025 IEEE International Radar Conference (RADAR)*, 2025, pp. 1–6. DOI: 10.1109/RADAR52380.2025.11031915.

[118] S. A. Flandermeyer and J. G. Metcalf, "Relational Reinforcement Learning for Autonomous Search and Track," *IEEE Transactions on Aerospace and Electronic Systems (Submitted)*,

[119] T. K. Rusch, M. M. Bronstein, and S. Mishra, *A survey on oversmoothing in graph neural networks*, 2023. arXiv: 2303.10993 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2303.10993.

[120] B. Zhang and R. Sennrich, "Root mean square layer normalization," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA: Curran Associates Inc., 2019.

[121] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, 2019, pp. 3744–3753.

[122] C. Liu, Y. Zhan, J. Wu, *et al.*, "Graph pooling for graph neural networks: Progress, challenges, and opportunities," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, ser. Ijcai '23, Macao, P.R.China, 2023, ISBN: 978-1-956792-03-4. DOI: 10.24963/ijcai.2023/752.

[123] C. Yang, L. Kaplan, and E. Blasch, "Performance Measures of Covariance and Information Matrices in Resource Management for Target State Estimation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 3, pp. 2594–2613, Jul. 2012, ISSN: 1557-9603. DOI: 10.1109/TAES.2012.6237611.

[124] A. S. Rahmathullah, Á. F. García-Fernández, and L. Svensson, "Generalized optimal sub-pattern assignment metric," in *2017 20th International Conference on Information Fusion (Fusion)*, Jul. 2017, pp. 1–8. DOI: 10.23919/ICIF.2017.8009645.

[125] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," *Computing*, vol. 38, no. 4, pp. 325–340, Nov. 1987, ISSN: 0010-485X. DOI: 10.1007/BF02278710.

[126] D. F. Crouse, "On implementing 2D rectangular assignment algorithms," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 4, pp. 1679–1696, Aug. 2016, ISSN: 1557-9603. DOI: 10.1109/TAES.2016.140952.

[127] D. F. Crouse, P. Willett, K. Pattipati, and L. Svensson, "A look at Gaussian mixture reduction algorithms," in *14th International Conference on Information Fusion*, Jul. 2011, pp. 1–8.

[128] S. D. Blunt and E. L. Mokole, "Overview of radar waveform diversity," *IEEE Aerospace and Electronic Systems Magazine*, vol. 31, no. 11, pp. 2–42, 2016. DOI: 10.1109/MAES.2016.160071.

[129] D. M. Rouse and S. S. Hemami, "Understanding and simplifying the structural similarity metric," in *2008 15th IEEE International Conference on Image Processing*, 2008, pp. 1188–1191. DOI: 10.1109/ICIP.2008.4711973.

[130] A. Jaegle, S. Borgeaud, J.-B. Alayrac, *et al.*, *Perceiver IO: A General Architecture for Structured Inputs & Outputs*, Mar. 2022. DOI: 10.48550/arXiv.2107.14795. arXiv: 2107.14795 [cs].

[131] G. Van Keuk and S. Blackman, "On phased-array radar tracking and parameter control," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 29, no. 1, pp. 186–194, Jan. 1993, ISSN: 1557-9603. DOI: 10.1109/7.249124.

Appendix A

Preliminary Graph Search and Track Formulation

A.1 Problem Formulation

This appendix presents a preliminary graph learning solution for the search and track problem that formed the foundation of the work in Chapter 4. Compared to this appendix, the approach in Chapter 4 significantly improves the graph representation, neural network architecture, and reward function to make the model highly scalable and general-purpose. However, the preliminary approach defined below is included for completeness and to motivate the generalized formulation. Much of the discussion in this chapter follows from the original conference publication [117].

The proposed formulation models relationships between objects in the environment (e.g., sensors and tracked objects) as a heterogeneous graph that is incrementally updated as the scenario evolves. The resulting graphs provide a concise state representation for a model-based RL agent that plans future actions in a learned world model without discretizing its control space. The graph RL approach is largely sensor-agnostic, scales to a time-varying number of tracked objects and search regions of interest, and supports massively parallel processing using a graph neural network. The performance of the agent is verified in a

mobile surveillance scenario and compared to a set of myopic heuristic policies.

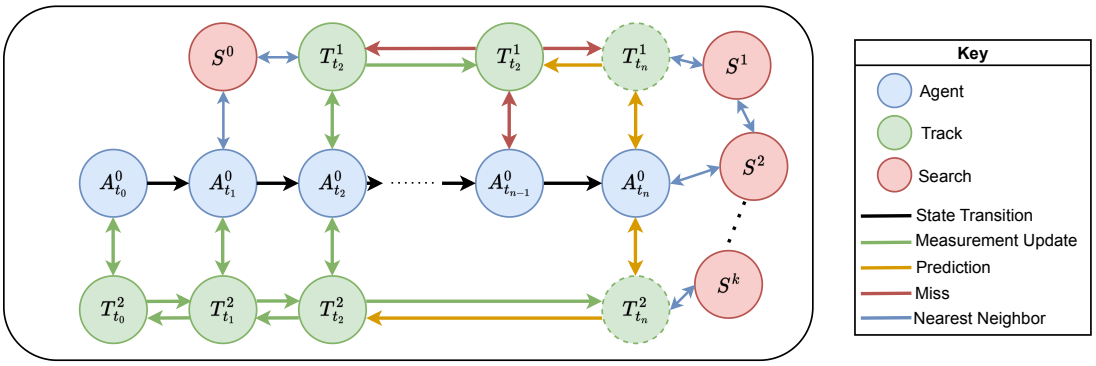
A.2 Proposed Algorithm

A.2.1 Graph State Representation

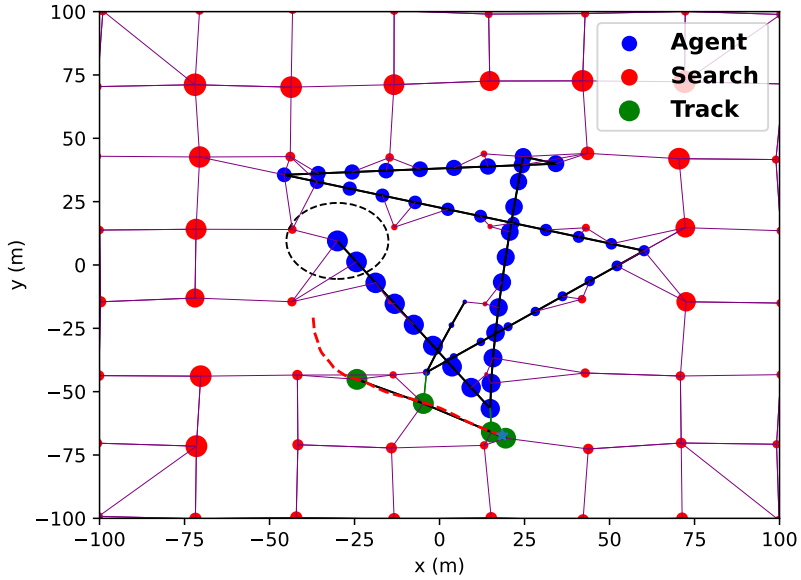
This section derives a graph-theoretic representation of the search-and-track environment history using the TOMB/P filter from Section 2.4. The environment state is a heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, U)$, where $\mathcal{V}$ and $\mathcal{E}$ are sets of nodes and edges, respectively (Figure A.1a), and U is a global feature vector. The node set $\mathcal{V}$ consist of agent, search, and track nodes. Agent nodes describe the state of the sensing platform at each time step. Search nodes describe components of the undetected object intensity λ^u, which is represented as a Gaussian mixture as in Equation (4.5). In general, increasing the number of mixture components results in a more finely-detailed search "grid" (which need not be uniformly spaced) at the expense of greater computation. Track nodes represent the state history of tracks in the RFS filter. Although the TOMB/P filter only maintains the current state estimate for each object, the graph representation stores track states over multiple time steps.

The edge set $\mathcal{E}$ comprises six five edge types (Figure A.1a):

- State transition edges: Connect agent nodes across subsequent time steps. These edges include relative pose information (e.g., translation and rotation) where required.
- Measurement update edges: Connect agent nodes to track nodes that were associated to a measurement in the same time step.
- Prediction edges: Connect the current agent node to the most recent node



(a) Abstract illustration of the graph representation



(b) Example graph from a simulated scenario

Figure A.1: Graph representation of the multi-object search-and-track environment. The scenario graph is composed of agent, search, and track nodes, which can be connected through various edge types. Search nodes act as spatial aggregators which share information between their k nearest neighbors.

for each track. The graph discards old prediction edges at each time step.

- Missed detection edges: Connect agent nodes to tracks in the same time step that have $p_d > 0$ but where not associated to any measurements.
- Nearest neighbor edges: Connect each search node to its k nearest spatial neighbors using the Mahalanobis distance as a metric [95].

Additionally, each track node is connected to the previous and subsequent nodes in its history with a predict/update/miss edge. Note that the nearest neighbor edge type introduces a computationally expensive pairwise distance calculation between search nodes, limiting the scalability of the approach when the search grid is large or high-resolution.

In practice, a decision-making agent rarely makes use of the entire scenario history. To reduce the size of the graph (and thus the memory/compute footprint of the algorithm), the environment observations only retain nodes and edges from the previous $T = 50$ time steps. Although it is not considered in depth here, future work could also examine Gaussian mixture reduction [127] strategies to further reduce computation requirements without sacrificing fidelity in the state representation.

A.2.2 Entity Features

Each node class in the heterogeneous graph possesses a unique set of features. Table A.1 provides a complete list of features for each class, where a checkmark indicates that the feature in question is included in the node representation. All nodes contain the kinematic state of their corresponding entity (i.e., the state vector from the tracker) and the node's age in units of time steps. This work assumes the agent state is perfectly known, so agent nodes do not include state

Table A.1: Node Features

Feature	Agent	Search	Track
Kinematics (position, velocity)	✓	✓	✓
Age (time steps)	✓	✓	✓
Covariance diagonals		✓	✓
Survival probability p_s		✓	✓
Log search weight		✓	
Existence probability r			✓
Covariance trace			✓

covariance or survival information. Track nodes include the Bernoulli existence probability r and survival probability p_s from the TOMB/P tracker. Since the agent actively estimates the state of tracked objects, track nodes also include the state covariance matrix and a track quality metric. In this case, the track quality measure is the covariance trace normalized by the maximum tolerable trace (see Section A.2.4). Edges also include feature vectors that describe relative kinematics (position, velocity) between nodes and a one-hot label indicating the edge class. Since the scale of undetected weights may vary significantly across scenarios, the weights are log-scaled before they are added to the search node feature vector. The feature vectors use the square root of covariance diagonals and traces, and the agent applies a symmetric logarithm function to all features in order to maintain a reasonable input scale [83].

A.2.3 Graph Encoder Architecture

This section presents a neural network architecture that produces a fixed-size vector representation of the environment from the dynamic graph state. Figure A.2 outlines the general network architecture. In the first step, node features are transformed independently using a 2-layer MLP which includes LayerNorm [51] and ReLU activations.

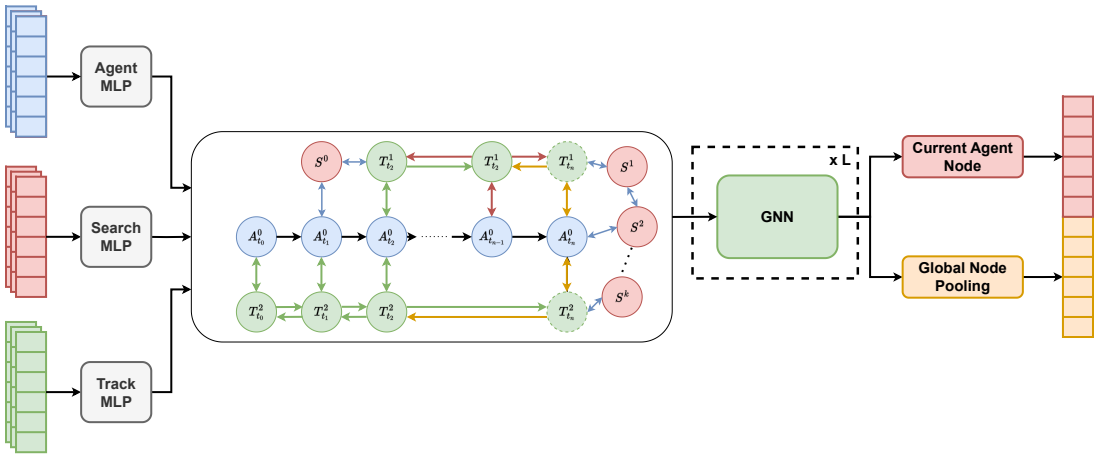


Figure A.2: General encoder architecture for the autonomous agent. Each node type is processed by a separate 2-layer MLP before forming the graph, and the graph is processed with L graph neural network (GNN) layers. The final feature vector is the vector concatenation of the current agent node and pooled node features.

Next, the encoder processes the graph formed via Section A.2.1 with an L-layer graph neural network to produce a latent representation of each node. The network processes the resulting node embeddings as a set, using a pooling by multi-head attention (PMA) layer to summarize the node states with a single vector [121]. The final output representation is the concatenation of the pooled representation and the *current* agent node's latent vector. The encoder output thus captures global context about the entire scenario history while retaining a precise representation of the local agent state.

Each of the $L = 2$ GNN layers in the results that follow uses the graph isomorphism network (GIN) update rule defined in Equation (2.16), where each layer uses a 2-layer MLP with LayerNorm and ReLU activations. The network incorporates edge features by projecting them to the embedding dimension and adding them element-wise to each neighbor embedding $h_u^{(k-1)}$ in the GIN update. The time complexity of the PMA and GIN layers is linear in the number of nodes,

making the architecture scalable to scenarios with dense surveillance grids and many tracked objects.

A.2.4 Reward Function

This section presents a reward function that encourages the agent to adaptively balance search and track performance. The reward function is a linear combination of task-specific components:

$$R_t = \alpha R_t^{track} + \beta R_t^{search}, \tag{A.1}$$

where R_t^{track} and R_t^{search} are the search and track rewards for time t, respectively, and α and β are scalar priority weights for each task. The following derivation defines task rewards such that they have similar scales for each task, simplifying the selection of α and β.

The search reward R_t^{search} is computed directly from the undetected intensity λ^u in the TOMB/P filter. As stated in Section A.2.1, λ^u is a Gaussian mixture whose component weights represent the expected number of undetected objects in a localized region of the state space. The filter update step scales these weights by the detection probability to reflect a reduction in uncertainty. To push the agent to obtain high-p_d observations on uncertain regions of the search volume, the search reward derives from the reduction in undetected intensity across time steps:

$$R_t^{search} = \frac{\sum_{i=0}^{N_{t|t-1}^u} w_{t|t-1}^{u,i} - \sum_{j=0}^{N_{t|t}^u} w_{t|t}^{u,j}}{\max(w_{t|t-1}^u)}, \tag{A.2}$$

where $w_{t|t-1}^u$ and $w_{t|t}^u$ are the predicted and posterior undetected mixture weights, respectively. Since $w_{t|t-1} \geq w_{t|t}$, the search reward is always non-negative. In

other words, the numerator measures the expected number of new objects "uncovered" by the most recent sensor action, and the denominator further constrains the reward contributions from each mixture component to the range $[0, 1]$.

The tracking task aims to maintain accurate state estimates on as many detected objects as possible. Though countless metrics for quantifying state estimation error exist [123], this work uses the trace of the state covariance matrix as a proxy for localization performance (where small traces indicate good state estimates). The track reward is the scaled one-step reduction in the total trace between all tracked objects:

$$R_t^{track} = \frac{\sum_{i=0}^{N_{t|t-1}^d} tr(P_{t|t-1}^i) - \sum_{j=0}^{N_{t|t}^d} tr(P_{t|t}^j)}{\max(tr(P_{t|t-1}))}, \tag{A.3}$$

where $P_{t|t-1}^i$ and $P_{t|t}^j$ are the ith and jth covariance matrices in the predicted and posterior Multi-Bernoulli distributions, respectively, and the $tr(\cdot)$ operator sums the diagonal terms of an input matrix. To minimize the impact of false track hypotheses (e.g., due to false alarms), only tracks with existence probability $r > r_{min}$ contribute to the reward. Like R^{search}, the track reward is also strictly non-negative. The denominator again constrains the reward contributions from each tracked objects to the range $[0, 1]$.

Since R^{search} and R^{track} have the same form and scale, the task priorities are set as a convex combination with $\alpha \in [0, 1]$ and $\beta = 1 - \alpha$. The tracking task priority is defined as follows:

$$\alpha = \frac{\sqrt{\max(tr(P_{t|t-1}))}}{\rho}, \tag{A.4}$$

where ρ is a user-specified maximum tolerable state estimation error. With

$\beta = 1 - \alpha$, Equation (A.4) encourages the agent to prioritize tracking when a track needs to be updated and search when all tracks have good state estimates. The behavior of the agent is thus entirely characterized by the tolerable error threshold ρ. Both Equations (A.3) and (A.4) clip the trace values to not exceed ρ^2. The complete reward function given by Equation (A.1) depends only on the outputs of the TOMB/P prediction and update steps.

A.3 Experimental Results

This section evaluates the performance of the graph RL approach in a 2D scenario where a mobile platform detects and tracks objects in a $100\,\text{m} \times 100\,\text{m}$ region of interest (Figure A.1b). The agent uses TD-MPC2 (Section 2.2) as its underlying RL engine, modeling sensor control as a continuous control problem that selects the platform velocity and heading at each time step. The platform is randomly relocated in the map at the start of each episode, collecting data at $\Delta t = 1\,\text{s}$ intervals for $1000\,\text{s}$ before the environment is reset. The agent moves with a maximum speed of $10\,\text{m/s}$ and detects objects within a $r_{max} = 15\,\text{m}$ radius according to the following detection function:

$$p_d(x_i) = \exp\left[-\frac{1}{2}\left(\frac{||p_i - p_{agent}||}{r_{max}}\right)^2\right], \tag{A.5}$$

where p_i and p_{agent} are the xy positions of the object and agent, respectively. The sensor provides position-only measurements of objects using a linear-Gaussian model that has a diagonal covariance with an error of $\sigma_p = 0.1\,\text{m}$ in each measured dimension. This simple scenario highlights the agent's ability to reason over long time horizons since the agent must perform path planning over many time steps to reach desirable states.

All non-platform object motion follows a nearly constant velocity (NCV) model [95] with process noise $\sigma_w^2 = 0.01\,\mathrm{m/s^2}$. The object birth model is a Gaussian mixture $\sum_{i=1}^{N^b} w^{b,i}\mathcal{N}(x;\mu^{b,i},P^{b,i})$ with $N^b = 49$ components. The birth weights $w^{b,i}$ are equal and set such that a new object is expected every 100 s. The mixture mean vectors $\mu^{b,i} = [x^{b,i}, 0, y^{b,i}, 0]$ form a 7×7 uniform grid in position, and the mixture covariance matrices are equal and defined as $P^{b,i} = \mathrm{diag}([100/7, 1, 100/7, 1])^2$ to represent a diffuse prior. To further diversify the training scenarios, the agent begins each episode with 0-3 pre-initialized tracks of varying quality and the initial number of undetected objects is uniformly initialized between 0 and 3.

A.3.1 Search-Only Agent

The first experiment considers the degenerate case of a search-only agent obtained by setting $\rho = \infty$ in Equations (A.1) and (A.4). This configuration encourages the agent to obtain high-p_d observations on undetected regions in λ^u with large weights while neglecting the tracking task entirely. The RL agent is compared to a baseline search-while-track policy that constantly drives the agent to illuminate the undetected intensity component with the largest weight.

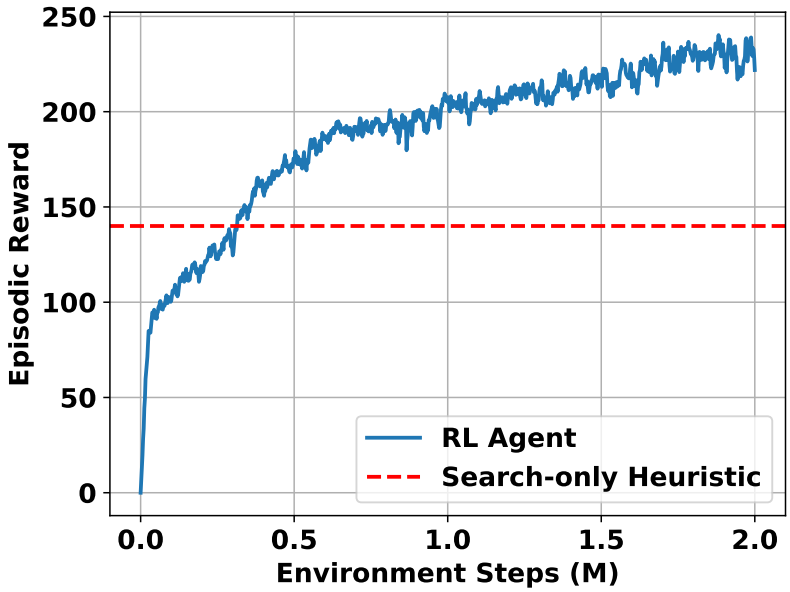
Figure A.3a shows the reward obtained by the agent during the training interval. The red dashed line indicates the average reward from the search-while-track heuristic over 1000 Monte Carlo episodes. The RL agent learns to outperform the heuristic policy in under 300k time steps and converges to an average reward of approximately 230 after 2M steps. The smooth convergence of the agent verifies that the graph encoder produces a useful state representation for learning the task. Though the heuristic gives reasonable results compared to the agent at the start of training, the path planning capabilities of the RL

agent are indispensable and provide a clear advantage over the greedy policy. Figure A.4 more thoroughly characterizes the tracking performance (localization error) and search performance (total undetected weight) of the RL agent for the search-only case. The next section describes this performance in more detail and relates it to an agent trained to search *and* track.

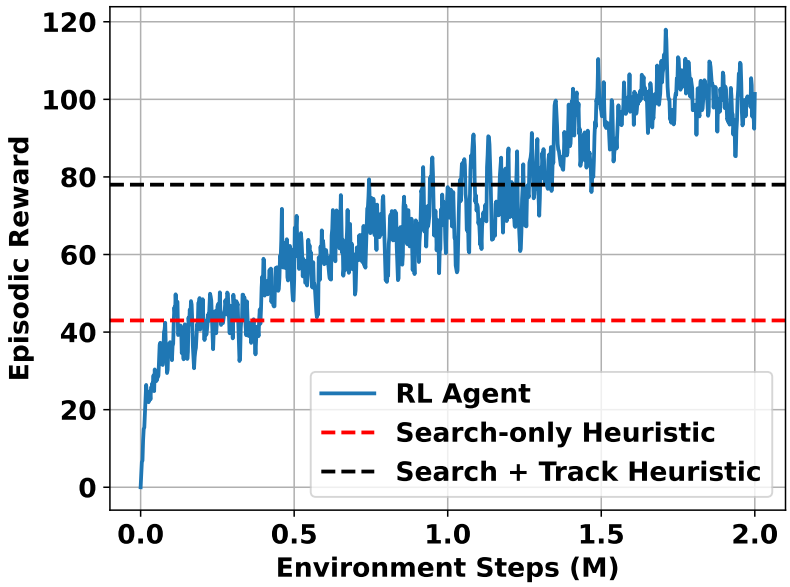
A.3.2 Joint Search-and-Track Agent

The next experiment evaluates an RL agent trained on the joint reward function with error tolerance $\rho = 15\,\text{m}$ (equal to the detection range of the sensor). The experiment compares the agent to a heuristic which illuminates the search region with the highest weight as before, but interrupts its search to update tracks once their localization error exceeds 0.5ρ. This behavior is inspired by traditional active radar trackers that optimize their dwell schedules to meet a track sharpness constant relative to the radar beam [131]. This experiment also adds clutter measurements which are uniform in the sensor field-of-view and Poisson-distributed at a rate of 1 false alarm per time step. Only the xy position components of the covariance are used to compute the traces in Equation (A.3) and in the performance metrics of Figure A.4.

Figure A.3b shows the reward obtained by the RL agent over the training period, along with the reward of the joint search-and-track heuristic and the search-only policy from the previous section. The dashed lines again indicate the average reward for each heuristic over 1000 Monte Carlo trials. Unsurprisingly, the RL agent and joint heuristic significantly outperform the search-only policy when tracking constraints are added to the reward, obtaining rewards of 79 and 100 on average, respectively. Overall reward values in this scenario are lower because the tracking task "steals" reward from the search task due to the scale

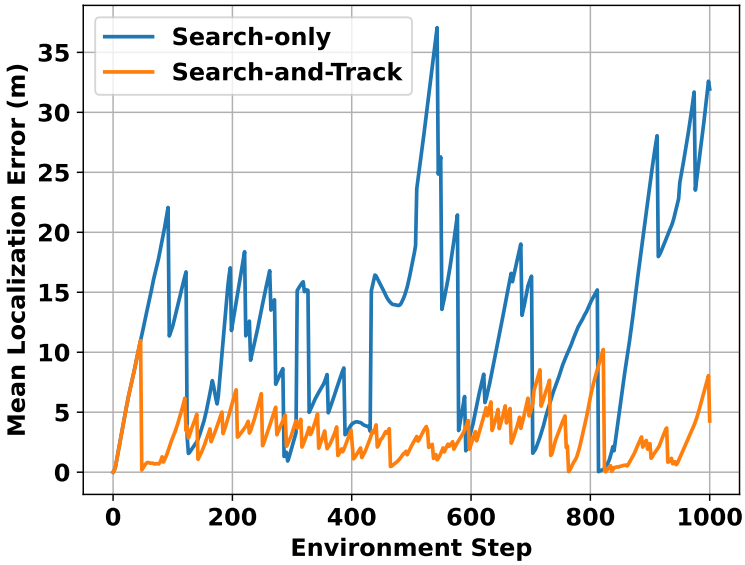


(a) Search-only mode ($\rho = \infty$)

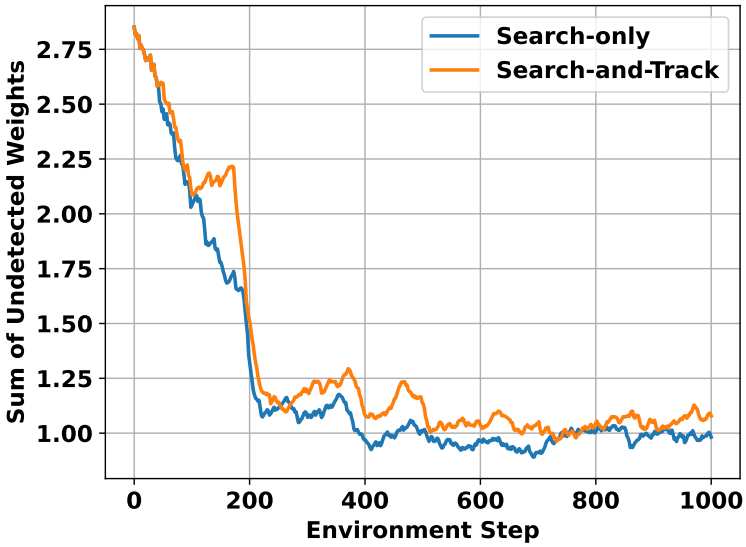


(b) Joint search-and-track mode ($\rho = 15\,\mathrm{m}$)

Figure A.3: Episodic training reward for RL agents with (a) no track maintenance requirements and (b) a maximum tolerable localization error of 15 m. In both cases, the RL agent learns to outperform its heuristic counterparts.



(a) Mean localization error



(b) Expected number of undetected objects

Figure A.4: Search and track metrics for the RL agents over a representative episode. The search-and-track agent maintains significantly lower localization error than the search-only agent with minimal impact to the search performance.

factor defined by Equation (A.4). The RL agent outperforms the joint heuristic after approximately 1.25M time steps, though the performance gap is less significant than in Figure A.3a.

To further characterize the learned behavior of the RL agents, Figure A.4 plots the two primary metrics for track and search over a representative episode in the environment simulation. Figure A.4a shows the localization error over time for both agents. The search-and-track agent maintains an average localization error of 3.9 m throughout the episode, which is well below the maximum 15 m tolerance. Interestingly, the search-and-track agent tends to avoid localization errors above 10 m so that it does not lose tracks entirely. By contrast, the search-only agent's average localization error is 14.7 m and its mean tracking error frequently exceeds the 15 m threshold.

Figure A.4b shows the sum of undetected intensity weights over the same episode, which provides an approximate measure of the number of undetected objects in the entire map. Both agents consistently decrease the search uncertainty over time, with the search-only and search-and-track agents obtaining average total weights values of 1.2 and 1.3, respectively, over the entire episode. Though the search-only agent consistently outperforms the search-and-track agent in this metric, the difference is relatively small. This indicates that the search-and-track agent learns to efficiently incorporate a surveillance pattern into its track update behavior.

A.4 Summary

This appendix presented a novel reinforcement learning solution for the joint search-track problem. The approach introduced a graphical representation of the environment history that follows directly from the output of the TOMB/P

filter. This work also introduced a scalable graph neural network architecture that produces a fixed-size latent vector from a dynamic environment state graph. The appendix also derives a novel reward function that adaptively manages the trade-off between search and track tasks. The performance of the graph learning agent was verified in a 2D mobile surveillance scenario, showing that the RL agent outperforms a set of reasonable baseline policies.

Appendix B

Acronyms and Abbreviations

BPTT	Backpropagation through time
CE	Cross-entropy
CNN	Convolutional neural network
CPI	Coherent processing interval
CPU	Central processing unit
CT	Coordinated turn
CWNA	Continuous white noise acceleration
DA	Data association
DFT	Discrete Fourier transform
DDPG	Deep deterministic policy gradient
DDQN	Double deep Q-Network
DQN	Deep Q-Network
DRQN	Deep recurrent Q-Network
DSA	Dynamic spectrum access
EKF	Extended Kalman filter
FFN	Feedforward neural network
FFT	Fast Fourier transform

FOV	Field of view
FPGA	Field-programmable gate array
FSS	Fast spectrum sensing
HOCAE	Hardware-optimized cell averaging estimation
GAT	Graph attention network
GCN	Graph convolutional network
GIN	Graph isomorphism network
GNN	Graph neural network
GOSPA	Generalized optimal sub-pattern assignment
GPU	Graphics processing unit
GRU	Gated recurrent unit
ISL	Integrated sidelobe level
ISM	Industrial, scientific, and medical
LFM	Linear frequency modulated
LMB	Labeled multi-Bernoulli
LN	Layer normalization
LLM	Large language model
LSTM	Long short-term memory
MAP	Maximum a posteriori
MARL	Multi-agent reinforcement learning
MB	Multi-Bernoulli
MCTS	Monte Carlo tree search
MDP	Markov decision process
MHA	Multi-head attention
MLP	Multi-layer perceptron
MMSE	Minimum mean square error

MPC	Model predictive control
MPPI	Model predictive path integral
NCV	Nearly constant velocity
PAC	Perception-action cycle
pdf	Probability density function
PE	Positional encoding
PHD	Probability hypothesis density
PMA	Pooling by multi-head attention
PMBM	Poisson multi-Bernoulli mixture
pmf	Probability mass function
POMDP	Partially observable Markov decision process
PPO	Proximal policy optimization
PPP	Poisson point process
PRF	Pulse repetition frequency
PRI	Pulse repetition interval
PSF	Point spread function
PSL	Peak sidelobe level
PU	Primary user
RDM	Range-Doppler map
ReLU	Rectified linear unit
RF	Radio frequency
RFS	Random finite sets
R-GGCN	Recurrent gated graph convolutional network
RL	Reinforcement learning
RSM	Range sidelobe modulation
RNN	Recurrent neural network

SAA	Sense-and-avoid
SAC	Soft actor-critic
SDR	Software-defined radio
SINR	Signal-to-interference-plus-noise ratio
SNR	Signal-to-noise ratio
SLAM	Simultaneous localization and mapping
SU	Secondary user
SSIM	Structural similarity index measure
TD-MPC	Temporal-difference model predictive control
TD-MPPI	Temporal-difference model predictive path integral
TOMB/P	Track-oriented multi-Bernoulli/Poisson
TPU	Tensor processing unit
UAS	Unmanned aerial system
UKF	Unscented Kalman filter