SCALABLE LEARNING IN DISTRIBUTED ROBOT TEAMS

Ekaterina Igorevna Tolstaya

A DISSERTATION

in

Electrical and Systems Engineering

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2021

Supervisors of Dissertation

Alejandro Ribeiro, Professor of Electrical and Systems Engineering

---

Vijay Kumar, Nemirovsky Family Dean of Penn Engineering and Professor of Mechanical Engineering and Applied Mechanics

Graduate Group Chairperson

Victor Preciado, Associate Professor and Graduate Chair of Electrical and Systems Engineering

Dissertation Committee

George Pappas, UPS Foundation Professor and Chair of Electrical and Systems Engineering
Raquel Urtasun, Professor of Computer Science, University of Toronto

Scalable Learning in Distributed Robot Teams

To Brent, who reminds me that there is more to life than science.

Acknowledgments

This thesis would not have been possible without my advisors, Alejandro Ribeiro and Vijay Kumar. I cannot begin to express my thanks to Alejandro, who has taught me to think independently and fearlessly. Alejandro has been my most honest critic and my strongest supporter. I'm extremely grateful to Vijay for giving me the most insightful feedback, and for providing the opportunities for me to grow as a researcher. I would also like to thank my committee, George Pappas and Raquel Urtasun for their feedback that has shaped this thesis.

I'm deeply indebted to my mentors, James Paulos, Alec Koppel, and Ethan Stump, for spending countless hours teaching me and working with me step-by-step through the stages of the research process. I would also like to thank the mentors I've been lucky to work with in industry, Reza Mahjourian, Murilo Martins, and Ashish Kapoor, for encouraging me to be creative, and supporting me in bringing my ideas to life. I'm grateful to Landon Butler for working with me during the pandemic. Landon's creativity and hard work motivated me to stay on track. I want to thank my collaborators Dan Mox and Fernando Gama for their support. I'm so grateful for the wonderful friends I've made at Penn, Arbaaz, Clark, Vik, Alena, Kim, Luana, Maria, Zhiyang, Sarah, Shih-Ling, Heejin, Vinicius, Harshat, Juan, Luiz, Santiago, Miguel, James S., Dinesh, Ty, Rebecca, Kelsey, Mike W., Mickey, Steven Chen, Monroe, Wenxin, and others who cheered me on and helped me look forward to every conference, group meeting, happy hour, sweet Friday and AWEsome lunch.

I thank the mentors who encouraged me in research projects prior to graduate school, Dr. Gomez, Dr. Krishnaprasad, Dr. Ghodssi, Dr. Culver, Jack Stansbury, Mark Estep, Mark Curran, and Dr. Stone.

And most importantly, I'm extremely grateful to my dad, Igor, for encouraging my interest in computers and science from an early age; my mom, Lada, for teaching me how to adult; my brother, Alex, for always jumping first, my dog, Roxy, for letting me pretend that I'm in charge, and my partner Brent, for supporting me while pushing me out of my comfort zone.

Abstract

Scalable Learning in Distributed Robot Teams

Ekaterina Igorevna Tolstaya
Alejandro Ribeiro
Vijay Kumar

Mobile robots are already in use for mapping, agriculture, entertainment, and the delivery of goods and people. As robotic systems continue to become more affordable, large numbers of mobile robots may be deployed concurrently to accomplish tasks faster and more efficiently. Practical deployments of very large teams will require scalable algorithms to enable the distributed cooperation of autonomous agents. This thesis focuses on the three main algorithmic obstacles to the scalability of robot teams: coordination, control, and communication. To address these challenges, we design graph-based abstractions that allow us to apply Graph Neural Networks (GNNs).

First, a team of robots must continually coordinate to divide up mission requirements among all agents. We focus on the case studies of exploration and coverage to develop a spatial GNN controller that can coordinate a team of dozens of agents as they visit thousands of landmarks. A routing problem of this size is intractable for existing optimization-based approaches. Second, a robot in a team must be able to execute the trajectory that will accomplish its given sub-task. In large teams with high densities of robots, planning and execution of safe, collision-free trajectories requires the joint optimization over all agent trajectories, which may be impractical in large teams. We present two approaches to scalable control: a) a controller for flocking that uses delayed communication formalized via a GNN; and b) an inverse optimal planning method that learns from real air traffic data. Third, robot teams may need to operate in harsh environments without existing communication infrastructure, requiring the formation of ad-hoc networks to exchange information. Many algorithms for control of multi-robot teams operate under the assumption that low-latency, global state information necessary to coordinate agent actions can readily be disseminated among the team. Our approach leverages GNNs to control the connectivity within the ad-hoc network and to provide the data distribution infrastructure necessary for countless multi-robot algorithms.

Finally, this thesis develops a framework for distributed learning to be used when centralized information is unavailable during training. Our approach allows robots to train controllers independently and then share their experiences by composing multiple models represented in a Reproducing Kernel Hilbert Space.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mobile robots are already in use for mapping, agriculture, entertainment, and the delivery of goods and people. As robotic systems continue to become more affordable, multiple robots may be deployed concurrently to accomplish their joint missions faster or more efficiently. As the complexity of missions is scaled up further, the deployment of tens or hundreds of robots may be required. Large-scale swarms of robots could be deployed to provide on-demand wireless networks [4], perform rapid environmental mapping [5, 6], track targets [7], search after natural disasters [8, 9], or enable sensor coverage in communication-denied environments [10]. At moderate scales, up to a dozen agents, it may be possible to centralize an entire team's information and control, but practical deployments of very large teams require distributed execution and scalable algorithms.

This thesis develops the algorithmic tools necessary for a large team of autonomous robots to cooperate on a joint mission. We focus on *autonomous* robots that incorporate sensing, control, and communication capabilities on-board, without relying on a central authority to provide direct mission commands, beyond an initial specification of the task. We also assume that the team is *homogeneous*, with all robots having the same capabilities. Finally, we focus on *cooperative* applications, with all robots in the team making progress on a single mission criteria. This thesis focuses on three main algorithmic obstacles to the scalability of robot teams: coordination, control, and communication:

Coordination. A team of robots deployed to accomplish a mission such as exploration, mapping or search must coordinate to perform the task efficiently, dividing up the mission requirements among all agents. Typically, this is accomplished through centralized approaches that jointly optimize over all agents. Such

approaches suffer from an exponential increase in complexity with the number of robots and the mission requirements. Furthermore, due to communication limitations, a central agent may be unable to provide commands with a high frequency to all members of the team. It has long been known that finding optimal policies in these distributed settings is challenging [11]. We focus on team coordination for the applications of coverage and exploration in Part I.

Control. A robot in a team, given a sub-task by a coordinator, must be able to execute a trajectory that will accomplish this objective. In large teams with high densities of robots, planning and execution of safe, collision-free trajectories requires the joint optimization over all agent trajectories, which may be impractical in large teams. We present two approaches for distributed scalable control: a) a controller for flocking that uses delayed communication formalized via a GNN, presented in Chapter 3; and b) an inverse optimal planning method that learns from real air traffic data, presented in Chapter 4. Chapter 3 describes the implementation of a distributed controller that leverages delayed communication to allow agents to avoid collisions and control their velocities for the task of flocking. Each agent must use their local observations and communication with nearby peers to determine their next acceleration. Chapter 4 presents a framework that uses inverse optimal planning to imitate real air-traffic, resulting in the generation of safe and feasible trajectories. The learned cost functions are interpretable and can be compared to existing standard procedures. Our approaches to the problems of distributed control and planning are presented in Part II.

Communication. In applications such as search after natural disasters, robot teams may need to operate in harsh environments without existing communication infrastructure, requiring the formation of ad-hoc networks to exchange information. Furthermore, agent actions may take them out of direct communication range with a subset of the team so that packets must be relayed through intermediate nodes to reach their intended destination. Ad-hoc robot networks are implicitly decentralized. In spite of this, many algorithms for control of multi-robot teams operate under the assumption that low-latency, global state information necessary to coordinate agent actions can readily be disseminated among the team. While detail about how this data distribution task is accomplished is often scarce, its success is vital to the overall performance of the team. Our approach to the data distribution problem is presented in Part III.

To develop scalable learning architectures for decentralized teams, we focus on designing graph-based abstractions. We first identify semantic entities such as robots or locations in space as nodes in a graph, and then we represent the relationships between these entities as edges in the graph. For example, robots'

communication or control capabilities can be encoded as edges. Such graph-based abstractions allow us to apply Graph Neural Networks (GNNs), which are information processing architectures that operate on network data in a local and decentralized way, harnessing useful information through repeated exchanges along edges of a graph [12]. In this thesis, we develop several GNN architectures that allow for *distributed inference* based only on the information local to each robot. Distributed inference is the key capability of our approach because it allows a large-scale team of autonomous robots to coordinate on a common mission using only local information for coordination, communication and control.

The first three parts of this thesis develop the tools necessary for distributed control, but they rely on a centralized training paradigm in which observations from all agents must be obtained simultaneously and the commands for all agents can also be issued at the same time. This assumption can be unrealistic even at training time, especially in the presence of communication delays or a huge number of agents. To address these challenges, Part IV seeks to develop a framework that could enable *distributed learning*, in which agents can learn independently and then share their distilled experiences by composing trained controllers. This framework relies on function approximation in a Reproducing Kernel Hilbert Space (RKHS). Chapter 6 develops a reinforcement learning algorithm that can train a non-parametric function approximation in an RKHS. Chapter 7 explains how these trained controllers can be combined by examining each function's coverage of the input space. We present obstacle avoidance as an example of a application for this framework and provide real robot experiments to demonstrate the performance of the composed controller. While the majority of this thesis focuses on the development and application of Graph Neural Networks, the RKHS-based function parameterization presented in Part IV is not specific to GNNs and may be applied to other types of models.

A Graph Representations of Robot Teams

In Chapters 2, 3, and 5, we focus on various application of Graph Neural Networks to multi-robot systems. Graph Neural Networks (GNNs) are an increasingly popular tool for exploiting the known structure of any relational system [13]. GNNs have been used to learn heuristic solutions to a variety of multi-robot problems, such as path planning [14, 13, 15], exploration [16], and perimeter defense [17]. Related works in multi-agent learning for control in imitation [17] and reinforcement [18] settings address varying neighbor relationships

in communication but provide no mechanism for multi-hop information flow. Explicit multi-hop message passing between all team members allows offline [19] and incremental [20] training for group inference but incurs a superlinear growth in communications with team size. Scalable online training is achieved in [21] by viewing the team as a distributed neural network, but training and inference must cease once agents move and the network changes.

In this thesis, we represent teams of robots as graphs, with their capabilities encoded as edges in the graph. We focus on the robots' abilities to communicate with each other, perceive their environment and act on entities in their environment, as in Fig. 1.1. For the application of coverage in Chapter 2, we design a task graph in which edges represent robots' movement capabilities, and then we apply graph neural networks to learn decentralized controllers. For the application of flocking in Chapter 3, we use the fixed-range communication capabilities of robots to generate a time-varying communication graph as agents move. For the data distribution application in Chapter 5, agents learn to construct the communication graph by balancing network latency and transmission power. Next, we will now develop the notation and architectures for Graph

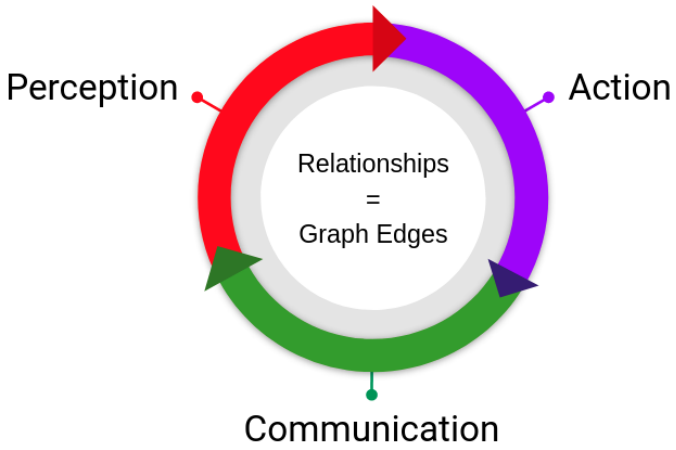


Figure 1.1. The capabilities of robots in a team can be encoded as edges in a graph, including the abilities of robots to communicate with each other, perceive their environment and act on entities in their environment.

Neural Networks (GNNs), the parametric models that can process these graph representations. We provide two different perspectives on Graph Neural Networks: the signal processing view and the relational view.

B The Signal Processing Perspective on Graph Neural Networks

Inspired by the development of Convolutional Neural Networks (CNNs) for images, we can define the Graph Convolution using learnable coefficients that multiply powers of the graph adjacency matrix times the graph signal [22, 12]. We rely on the graph adjacency matrix to capture the expected similarity between the signal components expressed at nodes. One powerful advantage of this formulation is that each layer of the GNN can apply multi-hop operations, which can be compared to using convolutional filters of sizes 2 pixels by 2 pixels or greater. Chapter 3 relies on the approach presented in this section.

Notation. For a matrix $\mathbf{A}$ with entries a_{ij} we use $[\mathbf{A}]_{ij} = a_{ij}$ to represent its (i, j)th entry and $[\mathbf{A}]_i = [a_{i1}, \ldots, a_{iN}]$ to represent its ith row. For matrices $\mathbf{X}$ and $\mathbf{Y}$ we use $[\mathbf{X}, \mathbf{Y}]$ to represent its block column concatenation and $[\mathbf{X}; \mathbf{Y}]$ for its block row stacking.

Control of Networked Systems. We consider a team of N agents distributed in space and tasked with controlling some large scale dynamical process. We characterize this dynamical process by the collection of state values $\mathbf{x}_i(t) \in \mathbb{R}^p$ observed at the locations of each agent i at time t, as well as the control actions $\mathbf{u}_i(t) \in \mathbb{R}^q$ that agents take. Grouping local states into the joint system state matrix $\mathbf{x}(t) = [\mathbf{x}_1^\mathsf{T}(t); \ldots; \mathbf{x}_N^\mathsf{T}(t)] \in \mathbb{R}^{N \times p}$, and local actions into the overall system action $\mathbf{u}(t) = [\mathbf{u}_1^\mathsf{T}(t); \ldots; \mathbf{u}_N^\mathsf{T}(t)] \in \mathbb{R}^{N \times q}$, we can write the evolution of the dynamical process through a differential equation of the form, $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t))$.

In order to design a controller for this dynamical system we operate in discrete time by introducing a sampling time T_s and a discrete time index n. We define $\mathbf{x}_n = \mathbf{x}(nT_s)$ as the discrete time state and $\mathbf{u}_n$ as the control action held from time nT_s until time $(n+1)T_s$. Solving the differential equation between times nT_s and $(n+1)T_s$, we end up with the discrete dynamical system

$$\mathbf{x}_{n+1} = \int_{nT_s}^{(n+1)T_s} f(\mathbf{x}(t), \mathbf{u}_n) \; dt + \mathbf{x}_n, \quad \text{with} \quad \mathbf{x}(nT_s) = \mathbf{x}_n. \tag{1.1}$$

At each point in (discrete) time, we consider a cost function $c(\mathbf{x}_n, \mathbf{u}_n)$. The objective of the control system is to choose actions $\mathbf{u}_n$ that reduce the accumulated cost $\sum_{n=0}^{\infty} c(\mathbf{x}_n, \mathbf{u}_n)$. When the collection of state observations $\mathbf{x}_n = [\mathbf{x}_{1n}^\mathsf{T}; \ldots; \mathbf{x}_{Nn}^\mathsf{T}]$ is available at a central location it is possible for us to consider centralized policies π that choose control actions $\mathbf{u}_n = \pi(\mathbf{x}_n)$ that depend on global information. In this case, the optimal policy is the one that minimizes the expected long term cost. If the dynamics in $f(\mathbf{x}(t), \mathbf{u}(t))$ and the costs $c\big(\mathbf{x}_n, \pi(\mathbf{x}_n)\big)$

are known, as we assume here, there are several techniques to find the optimal policy π^* [23, 24]. In this work, we are interested in decentralized controllers that operate without access to global information and interpret the optimal controller as a benchmark that decentralized controllers are trying to imitate.

The agent network is described by the presence of an edge $(i, j) \in \mathcal{E}_n$ which indicates that j may send data to i at time n. When this happens we say that j is a neighbor of i and define the neighborhood of i at time n as the collection of all its neighbors, $\mathcal{N}_{in} = \{j : (i, j) \in \mathcal{E}_n\}$. We can also define multi-hop neighborhoods of a node. We first define the 0-hop neighborhood of i to be the node itself, $\mathcal{N}_{in}^0 = \{i\}$. The 1-hop neighborhood of i is defined as simply the neighborhood of i, $\mathcal{N}_{in}^1 = \mathcal{N}_{in}$ We can now define the k-hop neighborhood of i as the set of nodes that can reach node i in exactly k hops. The k-hop neighbors of i are the nodes that are $(k-1)$-hop neighbors at time $n-1$ of the neighbors of i at time n. This recursive neighborhood definition characterizes the information that is available to node i at time n. This information includes the local state $\mathbf{x}_{in}$ that can be directly observed by node i at time n as well as the value of the state $\mathbf{x}_{j(n-1)}$ at time $(n-1)$ for all nodes j that are 1-hop neighbors of i at time n since this information can be communicated to node i. Node i can also learn the state $\mathbf{x}_{j(n-2)}$ of 2-hop neighbors at time $n-2$ since that information can be relayed from neighbors. In general, Eqn. 1.2 defines the information history $\mathcal{H}_{in}$ of node i at time n as the collection of state observations out to a maximal history depth K.

$$\mathcal{N}_{in}^k = \left\{ j' \in \mathcal{N}_{j(n-1)}^{k-1} : j \in \mathcal{N}_{in} \right\}, \qquad \mathcal{H}_{in} = \bigcup_{k=0}^{K-1} \left\{ \mathbf{x}_{j(n-k)} : j \in \mathcal{N}_{in}^k \right\} \tag{1.2}$$

The decentralized control problem consists on finding a policy that minimizes the long term cost $\sum_{n=0}^{\infty} c(\mathbf{x}_n, \mathbf{u}_n)$ restricted to the information structure in (1.2). This leads to problems in which finding optimal controllers is famously difficult to solve [11] except in some particularly simple scenarios [25]. This complexity motivates the introduction of a method that learns to mimic the global controller. Formally, we introduce a parameterized policy $\pi(\mathcal{H}_{in}, \mathbf{H})$ that maps local information histories $\mathcal{H}_{in}$ to local actions $\mathbf{u}_{in} = \pi(\mathcal{H}_{in}, \mathbf{H})$ as well as a loss function $\mathcal{L}(\pi, \pi^*)$ to measure the difference between the optimal centralized policy π^* and a system where all agents (locally) execute the (local) policy $\pi(\mathcal{H}_{in}, \mathbf{H})$. Our goal is to find the tensor of parameter $\mathbf{H}$ that solves the optimization problem

$$\mathbf{H}^* = \underset{\mathbf{H}}{\operatorname{argmin}} \, \mathbb{E}^{\pi^*} \left[\mathcal{L}\Big(\pi\big(\mathcal{H}_{in}, \mathbf{H}\big), \pi^*(\mathbf{x}_n) \Big) \right], \tag{1.3}$$

where we use the notation $\mathbb{E}^{\pi^*}$ to emphasize that the distribution of observed states $\mathbf{x}_n$ over which we compare

the policies $\pi\big(\mathcal{H}_{in}, \mathbf{H}\big)$ and $\pi^*(\mathbf{x}_n)$ is that of a system that follows the optimal policy π^*. The formulation in (1.3) is one in which we want to learn a policy $\pi\big(\mathcal{H}_{in}, \mathbf{H}\big)$ that mimics π^* to the extent that this is possible with the information that is available to each individual node. The success of this effort depends on the appropriate choice of the parameterization that determines the family of policies $\pi(\mathcal{H}_{in}, \mathbf{H})$ that can be represented by different choices of parameters, $\mathbf{H}$. In this work, we advocate the use of an aggregation graph neural network (Section B of Chapter 1) and demonstrate its applications to the problem of flocking with collision avoidance (Section A of Chapter 3).

Delayed Aggregation Graph Neural Networks. Aggregation graph neural networks (aggregation GNNs) are information processing architectures that operate on network data in a completely local and decentralized way, harnessing all the useful information by repeated exchanges with their neighbors [12]. This makes them suitable choices for parameterizing the decentralized policy $\pi(\mathcal{H}_{in}, \mathbf{H})$ in (1.3). However, aggregation GNNs operate on fixed graph signals defined over a fixed graph support, so in what follows, we extend aggregation GNNs to operate on time-varying graph processes defined over a time-varying graph support.

The graph support $\mathcal{G}_n$ is conveniently described by a *graph shift operator* $\mathbf{S}_n \in \mathbb{R}^{N \times N}$ which is a matrix whose element $[\mathbf{S}_n]_{ij}$ can be nonzero only if $(j,i) \in \mathcal{E}_n$ or if $i = j$, respecting the sparsity of the graph [26, 27, 28]. It follows that $\mathbf{S}_n$ defines linear, local operations since multiplications with $\mathbf{S}_n$ can be computed only with neighboring exchanges. More precisely, recall that $[\mathbf{x}_{(n-1)}]_j = \mathbf{x}^{\mathsf{T}}_{j(n-1)}$ denotes the state observed by node j at time $n-1$ and further distribute the product $\mathbf{S}_n\mathbf{x}_{n-1}$ so that $[\mathbf{S}_n\mathbf{x}_{n-1}]_i$ is associated with node i. Since $[\mathbf{S}_n]_{ij} \neq 0$ only if $(j,i) \in \mathcal{E}_n$ or if $i = j$ we can write

$$\Big[\mathbf{S}_n\mathbf{x}_{n-1}\Big]_i = \sum_{j=i, j\in\mathcal{N}_{in}} \Big[\mathbf{S}_n\Big]_{ij}\Big[\mathbf{x}_{n-1}\Big]_j. \tag{1.4}$$

Thus, node i can carry its part of the multiplication operation by receiving information from neighboring nodes. Examples of valid shift operators include weighted and unweighted adjacency matrices as well as weighted, unweighted, or normalized Laplacians. Aggregation GNNs leverage the locality of (1.4) to build a sequence of recursive k-hop neighborhood [Eq. (1.2)] aggregations to which a neural network can be applied. More precisely, consider a sequence of signals $\mathbf{y}_{kn} \in \mathbb{R}^{N \times p}$ that we define through the recursion

$$\mathbf{y}_{kn} = \mathbf{S}_n\mathbf{y}_{(k-1)(n-1)}, \tag{1.5}$$

with the initialization $\mathbf{y}_{0n} = \mathbf{x}_n$. If we fix the time n and consider increasing values of k, the recursion in (1.5) produces a sequence of signals where the first element is $\mathbf{y}_{0n} = \mathbf{x}_n$, the second element is $\mathbf{y}_{1n} = \mathbf{S}_n\mathbf{y}_{0(n-1)} = \mathbf{S}_n\mathbf{x}_{n-1}$, and, in general, the kth element is $\mathbf{y}_{kn} = (\mathbf{S}_n\mathbf{S}_{n-1}\dots\mathbf{S}_{n-k+1})\mathbf{x}_{n-k}$. Thus, (1.5) is modeling the diffusion of the state $\mathbf{x}_{n-k}$ through the sequence of time varying networks $\mathbf{S}_{n-k+1}$ through $\mathbf{S}_n$. This diffusion can be alternatively interpreted as an aggregation; at node i we can define an aggregation sequence of K elements as

$$\mathbf{z}_{in} = \Big[\big[\mathbf{y}_{0n}\big]_i ; \big[\mathbf{y}_{1n}\big]_i ; \dots ; \big[\mathbf{y}_{(K-1)n}\big]_i \Big]. \tag{1.6}$$

The first element of this sequence is $[\mathbf{y}_{0n}]_i = [\mathbf{x}_n]_i = \mathbf{x}_{in}^{\mathsf{T}}$ which represents the local state of node i. The second element of this sequence is $[\mathbf{y}_{1n}]_i = [\mathbf{S}_n\mathbf{x}_{n-1}]_i$ which aggregates the states $\mathbf{x}_{j(n-1)}$ of 1-hop neighboring nodes $j \in \mathcal{N}_{in}^1$ observed at time $n-1$ with a weighted average. In fact, this element is precisely the outcome of the local average shown in (1.4). If we now focus on the third element we see that $[\mathbf{y}_{2n}]_i = [\mathbf{S}_n\mathbf{S}_{n-1}\mathbf{x}_{n-2}]_i$ is an average of the states $\mathbf{x}_{j(n-2)}$ of 2-hop neighbors $j \in \mathcal{N}_{in}^2$ at time $n-2$. In general, the $k+1$st element of $\mathbf{z}_{in}$ is $[\mathbf{y}_{kn}]_i = [\mathbf{S}_n\mathbf{S}_{n-1}\dots\mathbf{S}_{n-k+1}\mathbf{x}_{n-k}]_i$ which is an average of the states $\mathbf{x}_{j(n-k)}$ of k-hop neighbors $j \in \mathcal{N}_{in}^k$ observed at time $n-k$. From this explanation we conclude that the sequence $\mathbf{z}_{in}$ is constructed with state information that is part of the local history $\mathcal{H}_{in}$ defined in (1.2) and therefore a valid basis for a decentralized controller. We highlight in equations (1.4) and (1.5) that agents forward the aggregation of their neighbors' information, rather than a list of neighbors' states, further along to multi-hop neighbors.

An important property of the aggregation sequence $\mathbf{z}_{in}$ is that it exhibits a regular temporal structure as it is made up of nested aggregation neighborhoods. This regular structure allows for the application of a regular convolutional neural network (CNN) [12] of depth L, where for each layer $\ell = 1, \dots, L$, we have

$$\mathbf{z}_{in}^{(\ell)} = \sigma^{(\ell)}(\mathbf{H}^{(\ell)}\mathbf{z}_{in}^{(\ell-1)}) \tag{1.7}$$

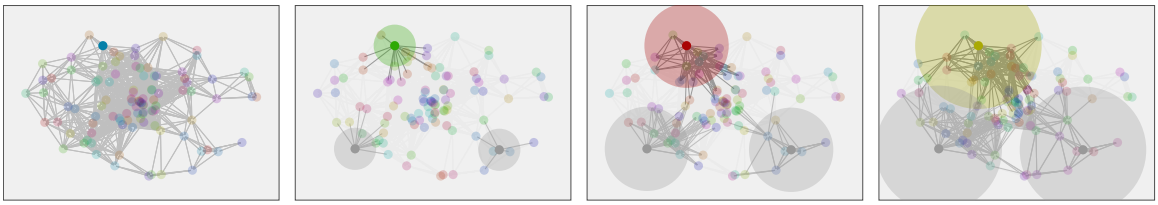
with $\sigma^{(\ell)}$ a pointwise nonlinearity and $\mathbf{H}^{(\ell)}$ a bank of small-support filters containing the *learnable* parameters. For each node i, we set $\mathbf{z}_{in}^{(0)} = \mathbf{z}_{in}$ and collect the output $\mathbf{z}_{in}^{(L)} = \mathbf{u}_{in}$ to be the decentralized control action at node i, at time n. We note that the filters $\mathbf{H}^{(\ell)}$ are shared across all nodes.

The aggregation GNN architecture, described in equations (1.5)-(1.7), constitutes a local parameterization of the policy $\pi(\mathcal{H}_{in}, \mathbf{H})$ that exploits the network structure and involves communication exchanges only with neighboring nodes. To *learn* the parameters for $\mathbf{H}$, we use a training set $\mathcal{T}$ consisting of sample trajectories

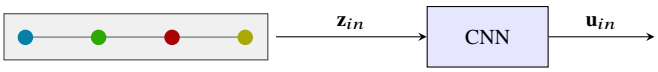
$(\mathbf{x}_n, \pi^*(\mathbf{x}_n))$ obtained from the global controller $\mathbf{u}_n^* = \pi^*(\mathbf{x}_n)$. We thus minimize the loss function over this training set, Eq. (1.3), where $\mathbf{u}_n$ collects the output $\mathbf{u}_{in} = \mathbf{z}_{in}^{(L)}$ of (1.7) at each node i:

$$\mathbf{H}^* = \underset{\mathbf{H}}{\operatorname{argmin}} \sum_{(\mathbf{x}_n, \pi^*(\mathbf{x}_n)) \in \mathcal{T}} \mathcal{L}\Big(\mathbf{u}_n, \mathbf{u}_n^*\Big) \tag{1.8}$$

We note that the policy learned from (1.8) can be extended to any network since the filters $\mathbf{H}^{(\ell)}$ can be applied independently of $\mathbf{S}_n$, facilitating transfer learning. This transfer is enabled by sharing the filter weights $\mathbf{H}$ among nodes at training time. The learned aggregation GNN models are, therefore, network and node independent. Graph covariance is an advantage for the applications examined in this work, but may not be a suitable assumption in all problems [29].



(a) $\mathbf{y}_{0n}$ (b) $\mathbf{y}_{1n}$ (c) $\mathbf{y}_{2n}$ (d) $\mathbf{y}_{3n}$



(e) Processing at the i-th node

Figure 1.2. Aggregation graph neural networks perform successive local exchanges between each node and its neighbors. For each k-hop neighborhood (illustrated by the increasing disks), record $\mathbf{y}_{kn}$ (1.5) to build signal $\mathbf{z}_n$ which exhibits a regular structure (1.6). Figures 1.2a The value of the state 1.2b One-hop neighborhood 1.2c Two-hop neighborhood 1.2d Three-hop neighborhood. 1.2e Once the regular time-structure signal $\mathbf{z}_n$ is obtained, we take each row $\mathbf{z}_{in}$, representing the information collected at node i, and we process it through a CNN to obtain the value of the decentralized controller $\mathbf{u}_{in}$ at node i at time n (1.7).

Aggregation GNN Evaluation. We present a detailed algorithm to compute the output of Aggregation GNNs, see Fig. 1.2. Essentially, each node in the network exchanges information with its neighbors repeatedly, building sequence $\mathbf{z}_{in}$ (1.6). Since this sequence is a regular sequence (i.e. nearby entries in $\mathbf{z}_{in}$ correspond to the information aggregated from neighboring nodes), a traditional CNN can be directly applied, and an output computed. This CNN is applied locally at each node, with parameters shared across all nodes.

More specifically, Algorithm 1 summarizes the inference methodology for the aggregation GNN at a single node of the network. At time n, the agent receive aggregation sequences from its neighbors $z_{j(n-1)}$. Then,

the agent pools this information from neighbors to form the current aggregation vector z_{in} which is input to the learned controller $\pi(\mathbf{z}_{in}, \mathbf{H})$ to compute the new action $\mathbf{u}_{in}$. Finally, the agent transmits its aggregation vector $\mathbf{z}_{in}$ to its current neighbors $\mathcal{N}_{in}$.

Algorithm 1 Aggregation Graph Neural Network at Agent i.

---

for n=0,1,. . . , **do**

Receive aggregation sequences from $j \in \mathcal{N}_{in}$ [eq. (1.6)]

$$\mathbf{z}_{j(n-1)} = \left[\left[\mathbf{y}_{0(n-1)} \right]_j ; \left[\mathbf{y}_{1(n-1)} \right]_j ; \ldots ; \left[\mathbf{y}_{(K-1)(n-1)} \right]_j \right]$$

Update aggregation sequence components [eq. (1.5) and (1.4)]

$$\left[\mathbf{y}_{kn} \right]_i = \left[\mathbf{S}_n \mathbf{y}_{k(n-1)} \right]_i = \sum_{j=1, j \in \mathcal{N}_{in}} \left[\mathbf{S}_n \right]_{ij} \left[\mathbf{y}_{k(n-1)} \right]_j$$

Observe system state $\mathbf{x}_{in}$

Update local aggregation sequence [cf. (1.6)]

$$\mathbf{z}_{in} = \left[\mathbf{x}_{in}^{\mathsf{T}} ; \left[\mathbf{y}_{1n} \right]_i ; \ldots ; \left[\mathbf{y}_{(K-1)n} \right]_i \right]$$

Compute next local action using the learned controller

$$\mathbf{u}_{in} = \pi(\mathbf{z}_{in}, \mathbf{H})$$

Transmit local aggregation sequence $\mathbf{z}_{in}$ to neighbors $j \in \mathcal{N}_{in}$

end for

C The Relational Perspective on Graph Neural Networks

In the previous section, we described an architecture that can process arbitrary graph signals whose structure is described by a given weighted adjacency matrix. Unfortunately, the complexity and diversity of the graph representation of a robot team often cannot be described by an adjacency matrix that has a single scalar edge weight for the relationship between each robot or entity. Furthermore, this weighting of edges may not be known a priori. For machine learning on heterogeneous graph representations, we require a more rich parametrization that can capture the algorithmic operations necessary for coordination, control and communication within a robot team.

In this section, we present an architecture that processes the graph as a set of relationships between discrete

entities in a multi-robot system, a framework first developed by [13]. The relational inductive bias of this architecture constrains the processing of information to occur only along the edges in the graph, which reduces the number of trainable parameters, as opposed to using fully connected networks. Furthermore, the prior of permutation invariance of nodes and edges is encoded in the structure of the aggregation functions that discard their ordering. These traits of relational architectures allow us to train rich models with non-linear learnable edge update functions, while preserving the ability of these models to generalize to the larger graphs that representing larger teams and more complex missions.

As compared to the architecture described in Section B, this formalism can only perform operations on immediate neighbors in the graph, equivalent to a 1-hop graph filter. This approach is very flexible in that it allows the application of non-linear graph updates as in 1.12. A linear graph network as in 1.11 is equivalent to a 1-hop graph filter. Chapters 2 and 5 leverage the approach from Section C.

This framework for Graph Neural Networks presented by DeepMind in [13] is designed to be extremely flexible and it unifies a variety of architectures, such as Graph Recurrent Neural Networks. In this section, we present the notation and language provided by the framework, along with our own novel architecture that is used in Chapters 2 and 5.

Following the approach of [13], the building block of a GNN is the Graph Network Block. Given a graph signal, $\mathcal{G} = \big\{\{\mathbf{e}_k\}, \{\mathbf{v}_i\}\big\}$, one application of the GN block transforms these features into $\mathcal{G}' = \big\{\{\mathbf{e}'_k\}, \{\mathbf{v}'_i\}\big\}$:

$$\mathbf{e}'_k = \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}), \quad \mathbf{v}'_i = \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i), \quad \bar{\mathbf{e}}'_i = \rho^{e\to v}(E'_i). \tag{1.9}$$

Note that this a simplification of the notation used by [13] since we have removed the ability to provide any inputs that are global to the system. $GN(\cdot)$ is a function of the graph signal $\mathcal{G}$, described by the application of ϕ^e, $\rho^{e\to v}$ and ϕ^v in that order to produce the transformed graph signal $\mathcal{G}'$, with the same connectivity but new features on the edges and nodes.

The aggregation operation $\rho^{e\to v}$ takes the set of transformed incident edge features $E'_i = \{\mathbf{e}'_k\}_{r_k=i}$ at node i and generates the fixed-size latent vector $\bar{\mathbf{e}}'_i$. Aggregations must satisfy a permutation invariance property since there is no fundamental ordering of edges in a graph. Also, this function must be able to handle graphs of varying degree, so the mean aggregation is particularly suitable to normalizing the output by the number

of input edges [30]:

$$\rho^{e\to v}(E_i') := \frac{1}{|E_i'|}\sum_{\mathbf{e}_k'\in E_i'}\mathbf{e}_k'. \tag{1.10}$$

The mean aggregation operation is especially helpful for improving the stability of GNNs with large receptive fields.

Next, we describe two novel variants of the Aggregation GNN architecture, and also first demonstrated in our work [31], and also detailed in Chapter 2. These two architectures also build upon the Agregation GNN architecture in [32]. The linear Aggregation GNN architecture uses the following parametrization:

$$\phi_L^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}) := \mathbf{v}_{s_k}, \quad \phi_L^v(\bar{\mathbf{e}}_i', \mathbf{v}_i) := \bar{\mathbf{e}}_i', \tag{1.11}$$

while the non-linear Aggregation GNN uses learnable non-linear functions to update node and edge features:

$$\phi_N^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}) := \mathrm{NN}_e([\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}]), \quad \phi_N^v(\bar{\mathbf{e}}_i', \mathbf{v}_i) := \mathrm{NN}_v([\bar{\mathbf{e}}_i', \mathbf{v}_i]), \tag{1.12}$$

where NN_e and NN_v are multi-layer perceptrons (MLPs). Note that the linear Aggregation GNN in (1.11) cannot use the input edge features, unlike the non-linear GNN defined in (1.12). While a Graph Network Block can be used to compose a variety of architectures, for this work, we develop a variant of the Aggregation GNN in which the output of every GN stage is concatenated, and finally, processed by a linear output transform f_{out} [31, 32].

$$\mathcal{G}' = f_{\text{out}}\Big(\big[f_{\text{dec}}(f_{\text{enc}}(\mathcal{G})), \quad f_{\text{dec}}(GN(f_{\text{enc}}(\mathcal{G}))), \quad f_{\text{dec}}(GN(GN(f_{\text{enc}}(\mathcal{G})))), \quad \ldots \quad \big]\Big) \tag{1.13}$$

The addition of the encoder f_{enc} and decoder f_{dec} layers was inspired by the Encode-Process-Decode architecture presented in [13]. Finally, f_{out} is a function that reduces the high-dimensional latent space vectors to the required low-dimensional output, such as 2D acceleration or logits of a Boltzmann distribution. The number of GN operations is a hyper parameter and determines the *receptive field* of the architecture, or how far information can travel along edges in the graph.

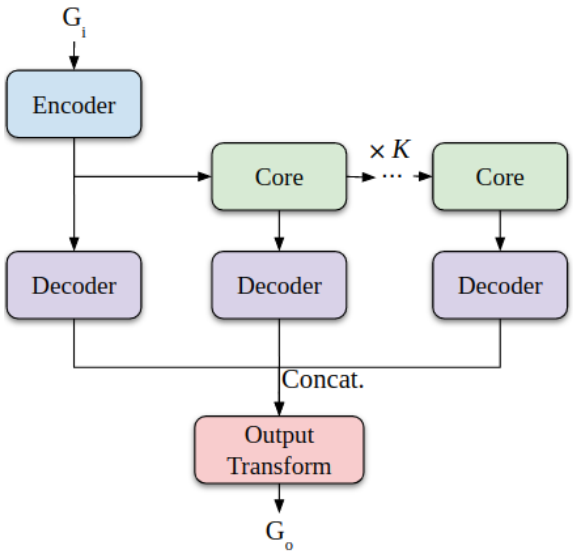


Figure 1.3. The Aggregation GNN architecture with G_i as the input graph signal and G_o as the output graph signal.

Part I

Coordination for Exploration and Coverage

Chapter 2

Coverage using Spatial Graph Neural Networks

Large scale swarms of robots could be deployed to provide on-demand wireless networks [4], perform rapid environmental mapping [5, 6], track targets [7], or search after natural disasters. At moderate scales, it may be possible to centralize the entire team's information and control in one agent, but practical deployments of very large teams require distributed execution and scalable algorithms. In particular, global planning-based approaches suffer from an exponential increase in complexity as the number of robots and the environment size increases. This motivates the use of heuristics in general and, as we advocate in this work, the use of learned heuristics.

Graph neural networks (GNNs) have been used to generate heuristic solutions to a variety of multi-robot problems, such as path planning [14, 13, 15], exploration [16], and perimeter defense [17]. In large teams, we can take advantage of graph equivariance to design abstractions that further speed up learning. In this work, we focus on the problem of coverage, in which a robot team must visit a set of locations in an environment [33]. To leverage recent advances in graph neural networks, we encode the task as a graph: the known map locations and team members are graph nodes, and allowed moves are graph edges. This approach allows us to abstract away the global agent and obstacle locations and to represent all elements of the problem in a single spatial graph with only local connectivity. Most importantly, the spatial graph representation can describe tasks with complex spatial constraints, unlike past work that applies GNNs to homogeneous inter-robot communication graphs [30, 34]. The development of the GNN architecture relies on Section C of Chapter 1.

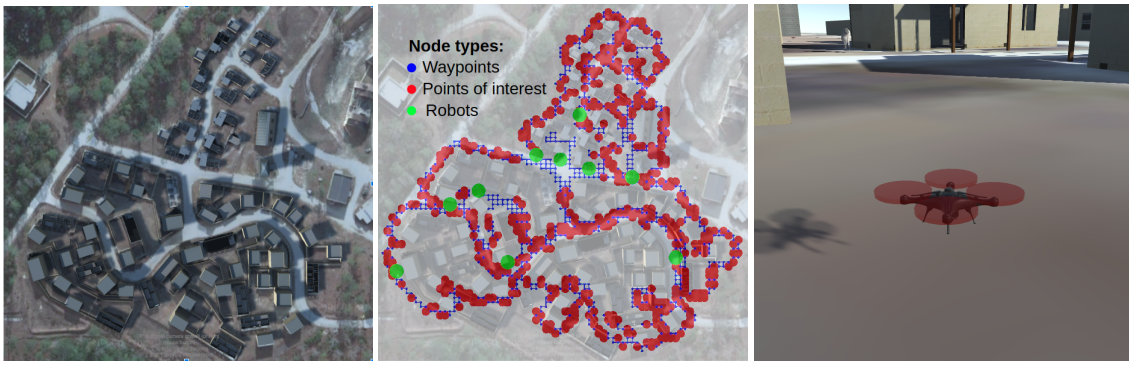
A moderate-size coverage task with dozens of goals and fewer than ten agents can be solved with existing

approaches when posed as a vehicle routing problem [35, 36]. We collect a dataset of trajectories generated using the centralized expert solution and use behavior cloning to train a graph neural network controller to imitate the expert solution. This learned heuristic can then generalize to previously unseen coverage scenarios with more agents and larger maps. We also show generalization to a scenario with simulated quadrotors that must traverse an environment with thousands of waypoints. Furthermore, we apply this approach to an exploration task, which is similar to the coverage task but the graph of waypoints is revealed to the robot team during mission execution. We demonstrate that by imitating an omniscient expert, our learned controller can outperform both a greedy controller and a receding horizon planner-based controller.

For the applications of coverage and exploration, information about distant points of interest may be necessary for computing the next position for each robot. To enable the learned controllers to use distant information, we build graph neural networks with a larger number of graph operation layers, up to 19 layers. The number of graph operation layers determines the distance along which information can travel from one node to another along the edges in the graph, the *receptive field* of the architecture. Other applications of GNNs to robot teams typically use receptive fields of 2 to 4 in conjunction with dense adjacency matrices [16, 34, 30, 37], with the exception of the shortest path demonstration in [13] with a receptive field of 10. A mean aggregation operation helps stabilize training of GNNs with larger receptive fields. Furthermore, we use a sparse representation of the local connectivity of the graph that allows our approach to scale to larger maps and teams than [37].

This chapter contributes methods and experimental validations for: (1) An approach to encoding continuous space multi-robot coverage and exploration problem data as a discrete spatial graph in which robots and points of interest are nodes and allowed moves are edges. (2) A training methodology using behavior cloning with an optimization based VRP expert solution for the coverage problem, as well as an extension to exploration with partially observable states. (3) A graph neural network architecture which explicitly respects equivariance in the task structure in order to achieve zero shot generalization to large maps and large teams for which an expert demonstration is intractable.

This chapter has been published as [31]. An open-source implementation of our work can be found at `https://github.com/katetolstaya/graph_rl`. A video demonstration is available at `https://youtu.be/MiYSeENTyoA`.



(a) A city simulated in Unity. (b) The graph representation of the task. (c) A team of 10 such quadrotors were used.

Figure 2.1. The trained models were tested on a team of robots simulated in Unity and controlled by waypoint commands issued through a Robot Operating System interface. The trained model allows the robots to divide and conquer to visit the points of interest more efficiently than a greedy model.

A Multi-Robot Routing for the Coverage Problem

We consider the time-constrained coverage path planning problem in which a team of robots must maximize the visited region of interest within a given time limit [33]. The problem of searching over the set of all feasible trajectories is intractable, so we finely discretize the region of interest, motivated by the success of motion planning in lattices [38]. In practice, a coarse map of a region may be provided via satellite imagery. To generate the lattice representation, we initialize a grid of points with a spacing of 5 meters, and then remove all points that are within an obstacle in the mission environment pictured in Fig. 2.1, and then add connections between adjacent nodes in free space.

We denote the set of robots as $\mathcal{R}$, waypoints as $\mathcal{W}$, and unvisited waypoints of interest denoted as $\mathcal{X}$. The set of unvisited waypoints of interest is a subset of all waypoints, $\mathcal{X} \subseteq \mathcal{W}$. Next, we define the parameters that describe the map of the environment: p^j is the location of waypoint j, and $\mathcal{N}_j$ is the set of neighboring waypoints to waypoint j. Then, we define the mission: $x_t^j \in \{0, 1\}$ is an indicator of whether the waypoint j is of interest at time t, with $x_t^j = 1$ if $j \in \mathcal{X}$. T denotes the time budget for the mission. The location of robot i at time t is q_t^i and $\mathbb{1}_{q_t^i = p^j}$ indicates whether robot i is currently at waypoint j. The multi-robot coverage

problem can now be formulated as:

$$\max_{\{q_t^i\}_{i\in\mathcal{R}}} \quad \sum_{t=0}^{T} \sum_{j\in\mathcal{W}} \sum_{i\in\mathcal{R}} x_t^j \mathbb{1}_{q_t^i=p^j} \tag{2.1}$$

$$\text{s.t. } x_t^j = x_0^j \prod_{i\in\mathcal{R}} \prod_{s=0}^{t-1} (1 - \mathbb{1}_{q_s^i=p^j}), \quad \forall j \in \mathcal{W}, t \leq T$$

$$q_{t-1}^i = p^j, q_t^i = p^k \Rightarrow k \in \mathcal{N}_j, \quad \forall i \in \mathcal{R}, \forall j, k \in \mathcal{W}, t \leq T.$$

Alternate formulations exist that can pose the coverage problem as a Vehicle Routing Problem (VRP) to be solved as a mixed-integer program [39]. For a given problem instance, we can use existing routing solvers such as [36] to generate open loop solutions. Other approaches to the coverage problem typically pre-compute the roles and trajectories of all team members for the duration of the mission [33].

In contrast, our goal is to develop a closed-loop controller that computes only the next action for each robot based on the current state of the system. This approach generalizes easily to systems with real dynamics in which agents may not instantaneously transition between desired waypoints, such as the team of simulated robots in Fig. 2.1. It also enables generalization to dynamic graphs, permitting us to extend the coverage problem to an exploration problem in which new waypoints are discovered online during execution. We seek to learn a closed loop controller, π, that maximizes the observed waypoints in expectation over the set of initial states and maps:

$$\max_{\pi} \quad \mathbb{E}_{q_0^i, p^j, x_0^j} \quad \sum_{t=0}^{T} \sum_{j\in\mathcal{W}} \sum_{i\in\mathcal{R}} x_t^j \ \mathbb{1}_{q_t^i=p^j} \tag{2.2}$$

$$\text{s.t. } x_t^j = x_0^j \prod_{i\in\mathcal{R}} \prod_{s=0}^{t-1} (1 - \mathbb{1}_{q_s^i=p^j}), \quad \forall j \in \mathcal{W}, t \leq T$$

$$q_{t-1}^i = p^j, q_t^i = p^k \Rightarrow k \in \mathcal{N}_j, \quad \forall i \in \mathcal{R}, \forall j, k \in \mathcal{W}, t \leq T$$

$$\{q_t^i\}_{i\in\mathcal{R}} = \pi\left(\{q_{t-1}^i\}_{i\in\mathcal{R}}, \ \{x_{t-1}^j\}_{j\in\mathcal{W}}, \ \{p^j\}_{j\in\mathcal{W}}\right).$$

B Methods

To solve the problem of multi-robot coverage, we develop a parametrization of the problem as a heterogeneous graph that can be input to a Graph Neural Network to be trained via supervised learning.

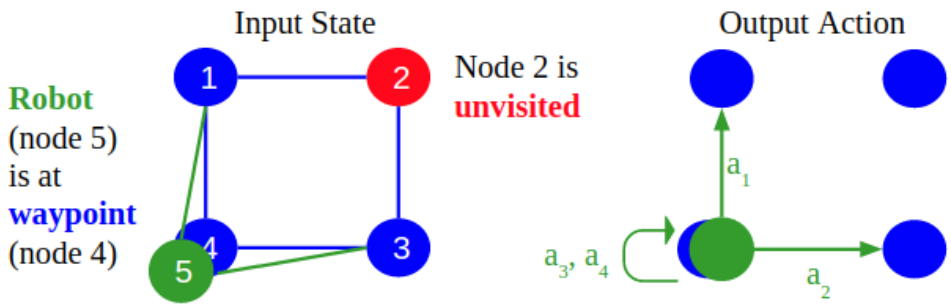


Figure 2.2. Robots and waypoints comprise the nodes in the graph, with the edges between them indicating the ability of robots to move to new locations.

Graph Representations for Coverage. We develop a parametrization of the current system state in which all entities, including robots and waypoints, are part of a single computation graph. Each robot in the team is considered as a node in the graph and robots are allowed to move from waypoint to waypoint in discrete steps. Due to robots' movement, the graph topology changes during execution. Using the lattice representation, we can abstract away the global positions of the robots, and maintain only relative information of nearby waypoints and robots. Unlike [16], we do not need to provide the relative distances of all waypoints to all agents. Furthermore, the action space of each robot is now discrete: the robot chooses one nearby waypoint to move to.

There are two types of edges: 1) map edges between waypoints to indicate free space and 2) action edges between robots and waypoints that indicate a robot's capability to move to nearby locations. Both types of edges are undirected. The waypoint connectivity is defined by the lattice induced over the given map of obstacles and free space. A robot is connected to the same waypoints as the waypoint at its current location, if $q_i = p_j$, then $\mathcal{N}_i = \mathcal{N}_j$.

The feature vector for each node $\mathbf{v}_i$ of index i indicates the type of this node:

$$\mathbf{v}_i = \{\mathbb{1}_{i\in\mathcal{R}}, \mathbb{1}_{i\in\mathcal{W}}, \mathbb{1}_{i\in\mathcal{X}}\}, \tag{2.3}$$

where $\mathbb{1}$ is an indicator function.

We define $\mathbf{e}_k$ as an edge feature vector for the directed edge of index k, with a sender node s_k and a

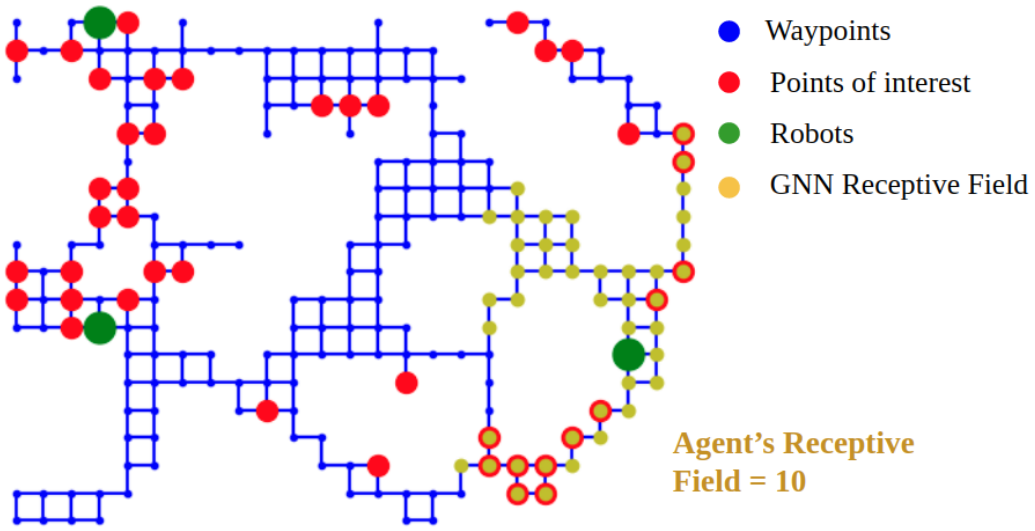


Figure 2.3. The receptive field (K=10) of a Graph Neural Network.

receiver node r_k. For this task, we define $\mathbf{e}_k$ to be the distance between the positions of nodes s_k and r_k:

$$\mathbf{e}_k = \|p_{s_k} - p_{r_k}\|. \tag{2.4}$$

The set of all edge features is $E = \{\mathbf{e}_k\}$, and the set of vertex features is $V = \{\mathbf{v}_i\}$, and we denote the graph that represents the state of the entire system as $G = \{E, V\}$. At a given time t, the state of the multi-robot task is described by G_t.

Graph Representations for Exploration. We view the exploration problem as the problem of coverage on a growing graph. Waypoint nodes are added to the graph when they are observed by a range sensor with range S: if $\|p_t^i - q_t^j\| \leq S$, then $\mathcal{W}_{t+1} = \mathcal{W}_t \cup \{p_i\}$, with the set of waypoints growing over time. Exploration introduces the possibility that an observed waypoint may or may not have adjacent waypoints that are currently unexplored. We call these frontier nodes and add an indicator feature to indicate whether a waypoint is part of the set of frontier nodes, $\mathcal{F}$:

$$\mathbf{v}_i = \{\mathbb{1}_{i\in\mathcal{R}}, \mathbb{1}_{i\in\mathcal{W}}, \mathbb{1}_{i\in\mathcal{X}}, \mathbb{1}_{i\in\mathcal{F}}\}. \tag{2.5}$$

Policy Architecture. For this application, we use the relational GNN architecture developed in Section C

of Chapter 1. We directly compare the linear and the non-linear architectures, developed in 1.11 and 1.12, respectively. For the non-linear model, the NN_e and NN_v are 3 layer MLPs with 16 hidden units. The architecture is shown in Fig. 1.3, with K indicating the number of aggregation stages in the architecture. The Core module is a Graph Network block, while the Encoder, Decoder, and Output Transform blocks process each node and edge attribute independently. For the non-linear model, the learnable parameters of the Core are shared among aggregation steps, and all learnable parameters are shared over all nodes and edges.

For both the linear and non-linear models, the Encoder encodes the local features of each node and edge into a larger latent space via a 3 layer MLP with no activation after the final layer. This is in contrast to the architecture in 3 that uses a hand-engineered feature extractor. The Decoder is also a 3 layer MLP with no final activation. The Output Transform is a linear layer that transforms the concatenated latent features to an output of dimension 1 for each node and edge. All layers used latent sizes of 16.

The resulting stochastic control policy is a function of the values output on the edges from each robot node to neighboring waypoints as shown in 2.2. The edge to move on is chosen using a Boltzmann distribution that uses a softmax action selection rule.

Fig. 2.3 visualizes a segment of a typical training scenario. Three robots shown in green must visit red points of interest by traveling along the waypoints in free space indicated by blue nodes and edges. We visualize the receptive field of a GNN with $K = 10$ indicating the 10-hop information available to the robot on the right. A larger receptive field would allow the agent to use information about more distant regions of the map to compute the controller. A GNN with a receptive field of zero can be compared to the Deep Set architecture that neglects any relational data [40].

Baseline Controllers. The learned policies are compared to three types of controllers: 1) an expert open loop VRP solution, 2) a receding horizon VRP-based controller, and 3) a greedy controller. We use Google's OR-Tools library [36] to provide optimization-based expert solutions to the VRP. This expert plans once for the full mission length of T and then this trajectory is executed in an open loop fashion. The expert assumes global knowledge of the map and may be intractable in larger or dynamic graphs. The training data was generated with this open-loop approach. We also devise a receding horizon controller that plans for trajectories of $\hat{T} < T$, and then executes the first step of this trajectory, and then re-plans at the next time step. The Expert controller baselines use receding horizon control in Figs. 2.4, 2.5, 2.9. Finally, a greedy controller that routes each robot to the nearest unvisited point of interest is a great heuristic in many scenarios, so we include it as

a practical lower bound. The greedy controller can be implemented with a finite receptive field using only a K-step distance matrix between nodes in the graph and we provide this benchmark in Figs. 2.4, 2.5, 2.9. A limited-horizon greedy controller may be more practical in larger graphs for which computing a full distance matrix is expensive.

Imitation Learning from Expert Demonstrations. In behavior cloning, our goal is to use stochastic gradient descent to minimize the difference between the expert's action and the policy's output, where $\mathcal{L}$ is a cross-entropy loss, since the action space is discrete:

$$\pi^* = \underset{\pi}{\operatorname{argmin}} \sum_{(G_t, \mathbf{u}_t) \in \mathcal{D}} \mathcal{L}\Big(\pi(G_t), \mathbf{u}_t\Big). \tag{2.6}$$

To use behavior cloning to train the graph neural network policy, we require a dataset of expert trajectories. We collect a dataset of 2000 expert trajectories of length $T = 50$ in randomly generated graphs, $\mathcal{D} = \{(G_t, \mathbf{u}_t)\}_{t=1,\ldots,50}$. The graphs are generated by sampling regions of 228 waypoints on average from the graph shown in 2.1. The performance of the learned controller is tested on graphs generated from the same distribution over trajectories of length $T = 50$. The models were trained for 200 epochs with a batch size of 32. Adam optimizer was used with an initial learning rate of 0.001 which was decayed by a factor of 0.95 for every 200 batches.

For the exploration task, we use the expert controller that uses the full graph to generate the trajectory, but only the local state observations are stored in the dataset, so the robot is learning to predict what the omniscient centralized controller would do based only on partial observations of currently explored nodes and frontiers.

Implementation Details. We use DeepMind's Graph Nets library and a variant of the Encode-Process-Decode architecture for the graph neural network policy [13].

We implement a local collision avoidance strategy as part of the task specification. A robot is allowed to move to a new waypoint if no other robot will be occupying it during the next time steps. Conflicts are resolved by giving priority to robots of smaller indices. In a real decentralized multi-robot team, local collision avoidance may also be necessary.

Due to the known maximum degree of the map graph, we can fix the number of neighboring waypoints considered by the policy to be up to 4, so that existing infrastructure for stochastic policies with fixed-size

action spaces can be used [41]. The output of the learned policy are weights for up to 4 edges from waypoints to robots, as shown in Fig. 2.2.

C Results

First, we highlight the impact of the Aggregation GNN's receptive field on its performance on the exploration and coverage tasks. Next, we examine how the GNN can generalize to larger graphs and team sizes. Finally, we validate the use of the GNN controllers in a high-fidelity simulator.

Locality. On the coverage task, the learned controllers reliably outperform a greedy controller, but fall short of the receding-horizon expert in Fig. 2.4. For the GNNs, we see a sharp improvement in performance with increasing receptive field. The linear and non-linear variants of the aggregation GNNs perform comparably, with the non-linear GNN performing slightly better. With an increasing horizon, the greedy controller rapidly reaches its asymptotic mean reward of 70.1 with a standard error of the mean (SEM) of 1.32. The open-loop expert obtains a mean reward of 91.0 with a SEM of 0.87.

On the exploration task, the learned controllers outperform greedy and the planner-based controllers as we can see in Fig. 2.5. Existing planning-based solutions cannot weigh the importance of a waypoint at the frontier versus other waypoints of interest, and we show that a learned solution can improve over a receding horizon expert controller. We also see a significant improvement in the mean reward obtained by the GNNs as their receptive field increases. The non-linear GNN is again slightly better than the linear GNN.

We further analyze the effect of a model's receptive field on its performance in varying size graphs. A finite receptive field decomposes the problem into local neighborhoods for each robot, producing a decentralized solution. The robot only uses a fixed-hop neighborhood of the graph for computing the action as seen in Fig. 2.3. The graph diameter is the max distance between any two nodes in a graph. A model with a smaller receptive field performs worse than a model with a larger receptive field, especially in graphs of larger diameter as demonstrated by Fig. 2.8. As the graph size increases, there are more points of interest that must be visited. The larger receptive field controller is able to route the agents to these waypoints, while the controller with the smaller receptive field ($K = 3$) is unable to compute a high-reward path.

While the expert controller outperforms the learned controllers in Fig. 2.4, the VRP-based controller requires an order of magnitude more time to compute a control action, as seen in Table 2.1. The neural

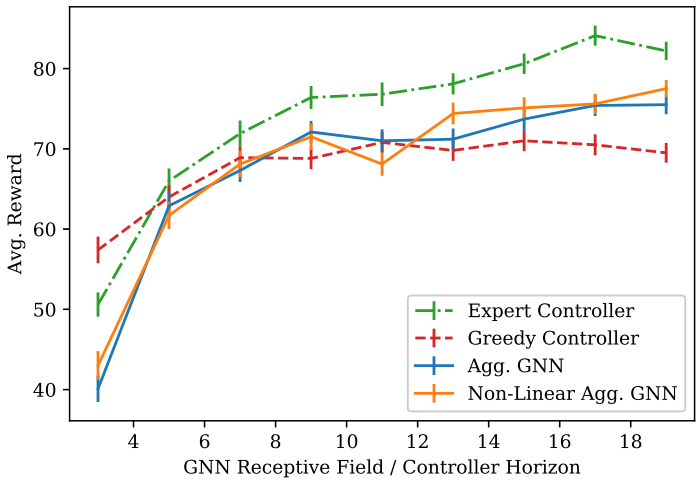


Figure 2.4. GNNs with larger receptive fields achieve higher rewards on the coverage task. Mean reward over 100 episodes with standard error is shown.

	Receptive Field		
Policy	K=9	K=19	∞
Expert	13 500	23 500	2330
GN-MLP	176	277	-
GN-Linear	133	171	-
Greedy	86.3	142	297

Table 2.1: The average controller computation time per episode in milliseconds, tested over 100 episodes of the coverage task. The expert controller (Receptive field = ∞) plans one open-loop trajectory for the task given the full adjacency matrix.

networks used a GTX 1080 GPU and the benchmarks used one core of an Intel Core i9-9900K processor. The expert and greedy controllers with receptive field of ∞ use the full adjacency matrix and, in particular, the expert plans one open-loop trajectory of length $T = 50$. We observe that the linear GNN has a faster controller computation time, and controllers with larger receptive fields also need more time.

Transference. The trained GNN models effectively generalize to larger robot team and map sizes. The models were first trained on 4 agents and 228 waypoints on average. Then, the models were tested on a map size of 5659 waypoints with a graph diameter of 205. The team size varied from 10 to 100 agents. For both the coverage and exploration generalization experiments, the map and team sizes made the centralized expert solution intractable. In Fig. 2.6, we observe that the learned solution consistently outperforms the

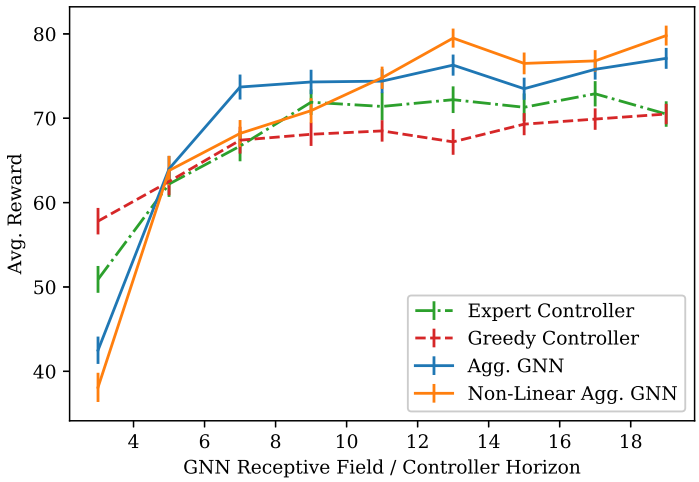


Figure 2.5. GNNs surpass the expert controller on the exploration task. Mean reward over 100 episodes with standard error is shown.

greedy controller on the coverage task. In Fig. 2.7, this difference is even bigger for the exploration task. We hypothesize that this is because the learned policy learns to weigh the frontier nodes more than other unexplored nodes.

Reinforcement Learning. Reinforcement Learning (RL) is also an effective tool for training Aggregation GNN controllers and achieves comparable performance to imitation learning in Fig. 2.9. RL is especially important in tasks that do not have suitable expert controllers for generation of training data. We use the Proximal Policy Optimization algorithm [42] and parametrize both the policy and value functions as Aggregation GNNs. The policy architecture is the same as for the imitation learning experiments, while the value function sums over the values output for the robot nodes. The GNN captures the known structure of the problem for the value and policy functions. The reward is the summation in the objective of (2.2), with no reward shaping required. The model was trained using 1×10^6 observations using an open source implementation of PPO from [41] using Adam optimizer with a step size of 1×10^{-4} and a batch size of 40.

Dynamics. Despite training on small-scale teams with instantaneous discrete state transitions, the GNN is effective for control in a coverage mission with ten quadrotors in a large simulated environment, pictured in Fig. 2.1. In 400 seconds of mission time, the team of 10 robots visited 490 points of interest using the greedy controller, as compared to visiting 610 points of interest using the non-linear GNN with a receptive field of 19.

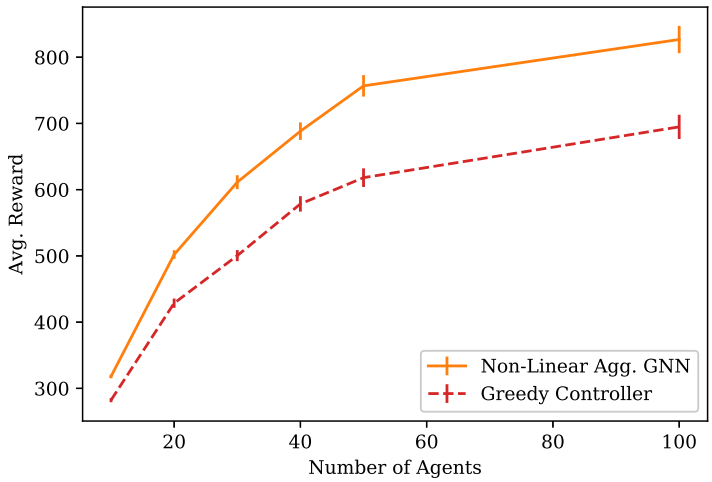


Figure 2.6. Generalization to a coverage task with 5659 waypoints. We plot the average reward over 100 episodes with standard error. The GNN was trained with 4 agents and tested on teams of up to 100.

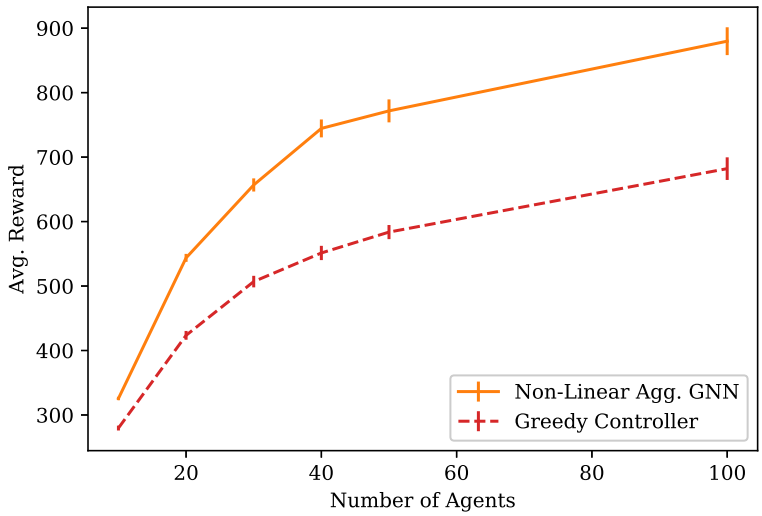


Figure 2.7. Generalization to an exploration task with 5659 waypoints. We plot the average reward over 100 episodes with standard error. The GNN was tested on teams of up to 100.

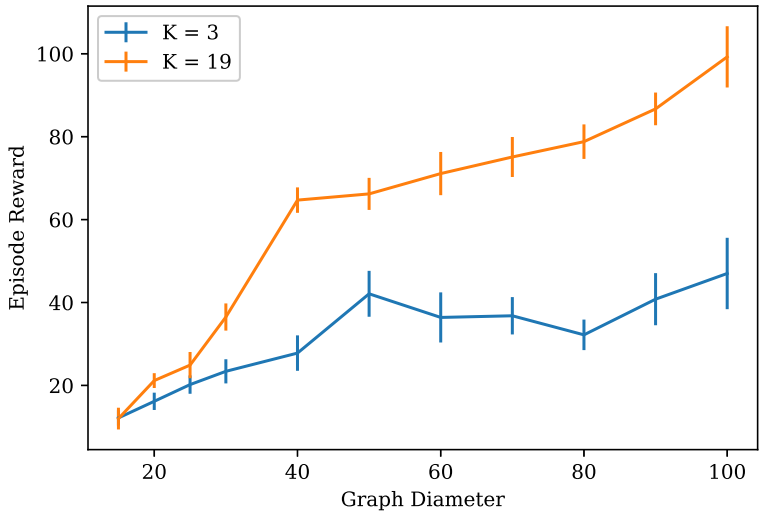


Figure 2.8. Effect of receptive field of non-linear GNNs in graphs of varying diameters, as measured by the mean reward over 20 episodes and standard error.

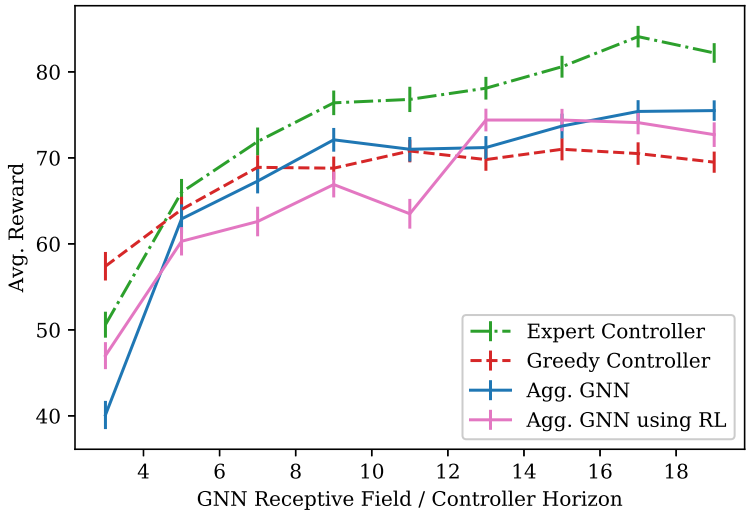


Figure 2.9. Comparison of linear GNNs trained with imitation learning and reinforcement learning. Mean reward over 100 episodes with standard error.

The use of the lattice representation with discrete states allows the model trained on an ideal discrete task to generalize zero-shot to a high-fidelity simulator. We discretize the robot positions provided by the simulator by clipping them to each robot's nearest waypoint. The spatial aggregation operations are invariant to the time-scale of the task, so we can execute the GNN controller with the non-ideal dynamics of a quadrotor.

D Summary

We develop a scalable GNN architecture for multi-robot coverage and exploration tasks. The approach surpasses existing decentralized heuristics and also scales well to from team sizes of 4 to teams of up to 100 agents. We also demonstrate that this architecture can be trained via reinforcement learning. As a bridge to deploying this approach to physical robot teams, we demonstrate generalization to a simulated robot team subject to dynamics in a dense urban environment.

To deploy the GNN in a real distributed team, we would need to address challenges such as asynchronous or intermittent communication. One possible solution could be the evaluation of the GNN on the contents of a robot's local buffer containing the estimated state of the system, and allowing intermittent communication among robots to update each other about the current positions of other robots, points of interest, and, for the exploration task, the growing map of waypoints.

Part II

Decentralized Control and Planning

Chapter 3

Flocking using Delayed Aggregation Graph Neural Networks

Large scale swarms are composed of multiple agents that collaborate to accomplish a task. For as long as scale is moderate, the group can be controlled as a whole from a central agent. However, as we reach for larger number of agents, decentralized control becomes a necessity. Individual agents must decide on control actions that are conducive to accomplishing a collective task from their local observations and communication with nearby peers. It has long been known that finding optimal controllers in these distributed settings is challenging [11]. This motivates the use of heuristics in general and, in particular and as we advocate in this chapter, the use of learned heuristics.

It is germane to emphasize that the challenge in decentralized control stems from the local information structure generated by the unavoidable restriction to communicate with only nearby agents. Building on this observation, we propose the use of imitation learning to train policies that respect the local information structure of a distributed system, while attempting to mimic the global policy of a clairvoyant central agent. The development of this architecture was provided in Section B of Chapter 1. The value of imitation learning has been demonstrated in single agent robotics problems such as autonomous driving [43] and quadrotor navigation [44]. When designing multi-agent systems, we must contend with the dimensionality growth of the system as new agents are added. Furthermore, it is unreasonable to assume that the network during training is the same as the network during execution.

Both of these problems can be overcome if we use Graph Neural Networks [45, 46, 22, 12, 47]. In particular, aggregation graph neural networks (aggregation GNNs) [12] are especially suited to teams of agents operating over a physical network because the architecture operates in an entirely local fashion involving only communication between nearby neighbors. In small-size multi-agent problems [48], it is shown that explicitly learning the graph of agent relationships aids in distilling decentralized agent policies from expert policies trained using Actor-Critic methods. In contrast, we leverage known relationships and connectivity between agents in order to extract features with graph convolutions, following the approach of [12]. Exploiting the known network structure allows us to consider teams an order of magnitude larger and highlights the value of using information from multi-hop neighbors.

Related work in multiagent learning for control in imitation [17] and reinforcement [18] settings address varying neighbor relationships in communication but provide no mechanism for multi-hop information flow. Explicit multi-hop message passing between all team members allows offline [19] and incremental [20] training for group inference but incurs a superlinear growth in communications with team size. Scalable online training is achieved in [21] by viewing the team as a distributed neural network, but training and inference must cease once agents move and the network changes.

We examine flocking tasks to highlight the ability of our approach to handle dynamic communication networks. Flocking has natural extensions to transportation and platooning of autonomous vehicles where agents rely on local observations to align their velocities and regulate their spacing [49]. Previous work focuses on developing local controllers which incorporate only observations from immediate neighbors [50, 51, 52]. We show that a global controller inspired by [52] outperforms such local controllers, but global approaches are not practical for real deployments. The novelty of our approach to flocking is to aggregate from multi-hop neighbors; this ability allows us to approach the performance of global solutions while respecting realistic communication constraints. Prior to this work, there has been no principled approach for augmenting the communication between neighbors to pass on information aggregated from multi-hop neighbors.

This chapter has been published as [30]. An open-source implementation of our work can be found at `https://github.com/katetolstaya/multiagent_gnn_policies`. A video demonstration is available at `https://youtu.be/Ph-GX0lSKME`.

A Methods: Learning to Flock

We evaluate the proposed methodology by learning a local controller for flocking with collision avoidance [52]. Following the signal processing-based approach developed in Section B of Chapter 1, we consider a team of N agents in which $\mathbf{r}_i$ denotes the position of agent i and $\mathbf{v}_i = \dot{\mathbf{r}}_i$ denotes its velocity. We assume that accelerations are fully controllable and therefore attempt to design control inputs $\mathbf{u}_i$ so that the change in velocity of agent i is $\dot{\mathbf{v}}_i = \mathbf{u}_i$. Further denote as $\mathbf{r}_{ij} := \mathbf{r}_j - \mathbf{r}_i$ the relative position of agent j with respect to agent i and introduce a constant $\rho > 1$ to define the collision avoidance potential

$$U(\mathbf{r}_i, \mathbf{r}_j) \;=\; 1/\|\mathbf{r}_{ij}\|^2 \;+\; \log\|\mathbf{r}_{ij}\|^2 \text{ if } \|\mathbf{r}_{ij}\| < \rho; \quad 1/\rho^2 + \log(\rho^2) \text{ otherwise.} \tag{3.1}$$

Since in addition to avoid collisions we want all agents to coordinate on their velocities we propose the controller

$$\mathbf{u}_i^* = -\sum_{j=1}^{N}(\mathbf{v}_i - \mathbf{v}_j) - \sum_{j=1}^{N}\nabla_{\mathbf{r}_i} U(\mathbf{r}_i, \mathbf{r}_j). \tag{3.2}$$

The combination of the collision avoidance potential with the velocity agreement term $\mathbf{v}_i - \mathbf{v}_j$ pushes the agents to coordinate their velocities while avoiding collisions. Observe that the potential $U(\mathbf{r}_i, \mathbf{r}_j)$ diverges at $\|r_{ij}\| = 0$, has a minimum when the distance between agents is $\|r_{ij}\| = 1$ and is indifferent when $\|r_{ij}\| > \rho$. It therefore pushes agents to either be at distance $\|r_{ij}\| = 1$ of each other or at distance $\|r_{ij}\| > \rho$ of each other.

The controller in (3.2) requires access to all agent velocities and all agent positions. In the parlance of Section B of Chapter 1, it is a clairvoyant centralized controller. In practice, agents can have access to local information only as they can only sense and communicate with agents within distance $\|\mathbf{r}_{ij}\| < R$ of each other. A controller that respects the locality of sensing and communication is

$$\mathbf{u}_i^\dagger = -\sum_{j\in\mathcal{N}_i}(\mathbf{v}_i - \mathbf{v}_j) - \sum_{j\in\mathcal{N}_i}\nabla_{\mathbf{r}_i} U(\mathbf{r}_i, \mathbf{r}_j). \tag{3.3}$$

The controller in (3.3) differs from the centralized controller in (3.2) in that the sums are over agents $j \in \mathcal{N}_i$ defined as those whose distance to agent i is $\|\mathbf{r}_{ij}\| < R$ for some communication radius R. For both the centralized and decentralized controllers, we have chosen $R = \rho$. It follows that the controller in (3.3) is decentralized as it can be implemented by accessing local information only. The controllers in (3.2) and (3.3)

have the same stationary points but (3.3) may indeed, it most often does, take longer to coordinate agent velocities. Next, we use the aggregation GNN described in Section B of Chapter 1 to learn a local controller that mimics (3.2). We will show that this learned controller outperforms (3.3) and achieves a performance that is similar to (3.2).

The global and local controllers are both non-linear in the states of the agents. The classical aggregation GNN approach described in Section B of Chapter 1 does not allow for non-linear operations prior to aggregation, so we cannot use the position and velocity vectors alone to imitate the global controller. Rather than directly using the state $[\mathbf{r}_i, \mathbf{v}_i]$ of each node i for aggregation, we design the relevant features needed to replicate the non-linear controller using only a linear aggregation operation, where $[\mathbf{x}_n]_i \in \mathbb{R}^6$:

$$[\mathbf{x}_n]_i = \left[\sum_{j \in \mathcal{N}_i} (\mathbf{v}_{i,n} - \mathbf{v}_{j,n}), \quad \sum_{j \in \mathcal{N}_i} \frac{\mathbf{r}_{ij,n}}{\left\|\mathbf{r}_{ij,n}\right\|^4}, \quad \sum_{j \in \mathcal{N}_i} \frac{\mathbf{r}_{ij,n}}{\left\|\mathbf{r}_{ij,n}\right\|^2} \right] \tag{3.4}$$

This observation vector is then used during aggregation as described in (1.5)-(1.6). The local controller also requires the computation of (3.4), so we are not giving the GNN an unfair advantage by providing the instantaneous measurements of neighbors' states. We quantify the cost of a trajectory by the variance in velocities, where T is the number of time steps in the trajectory. The variance in velocities measures how far the system is from consensus in velocities [53]:

$$C = \frac{1}{N} \sum_{n=1}^{T} \sum_{j=1}^{N} \left\| \mathbf{v}_{j,n} - \frac{1}{N} \left[\sum_{i=1}^{N} \mathbf{v}_{i,n} \right] \right\|^2 . \tag{3.5}$$

For our baseline scenario we consider a flock of $N = 100$ agents with a communication radius of $R = \rho = 1.0$ m and a discretization time period of $T_s = 0.01$ s. The flock locations were initialized uniformly on the disc with radius $\sqrt{N}$ to normalize the density of agents for changing flock sizes. Initial agent velocities are controlled by a parameter $v_{init} = 3.0$ m/s: agent velocities are sampled uniformly from the interval $[-v_{init}, +v_{init}]$ and then a bias for the whole flock is added, also sampled from $[-v_{init}, +v_{init}]$. To eliminate unsolvable cases, configurations are resampled if any agent fails to have at least two neighbors or if agents begin closer than 0.1 m. Finally, acceleration commands are saturated to the range $[-100, 100]$ m/s^2 to improve the numerical stability of training.

To obtain an estimate of the action to take, each agent operates an aggregation GNN architecture, as

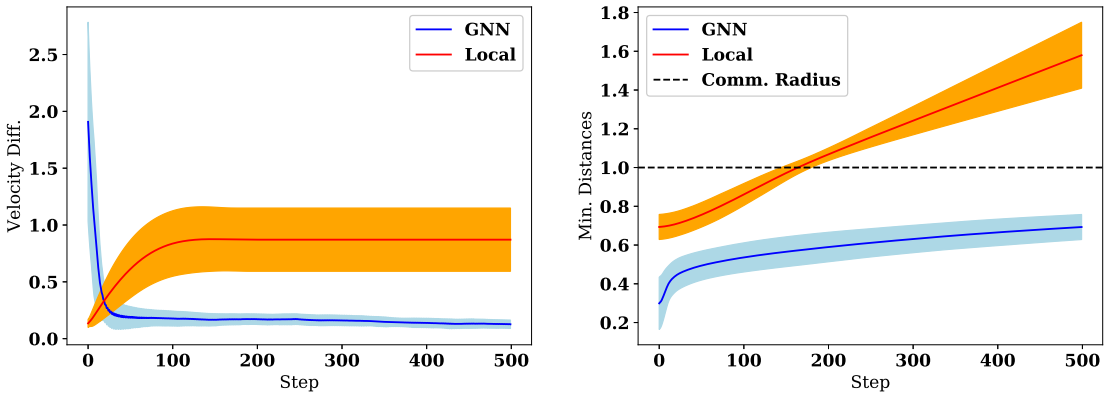
described in Section B of Chapter 1, with input features given by (3.4). The aggregated vectors $\mathbf{z}_{in}$ in Eq. (1.6) are built from $K-1$ neighbor exchanges, and then fed into a fully connected neural network as per Eq. (1.7), with two hidden layers of 32 neurons each and a Tanh activation function. The network was implemented in PyTorch and trained over a MSE cost function, using the Adam optimizer with learning rate $5 \cdot 10^{-5}$ and forgetting factors $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Each model was trained using 400 trajectories, each of length 200 steps total. For testing, 20 trajectories of length 200 were observed by using the learned controller only.

In practice, following the optimal policy to collect training data results in a distribution of states that is not representative of those seen at test time. To resolve this we use the Dataset Aggregation (DAgger) algorithm and follow the learner's policy instead of the expert's with probability $1-\beta$ when collecting training trajectories [54]. The probability β of choosing the expert action while training is decayed by a factor of 0.993 after each trajectory to a minimum of 0.5.

B Results

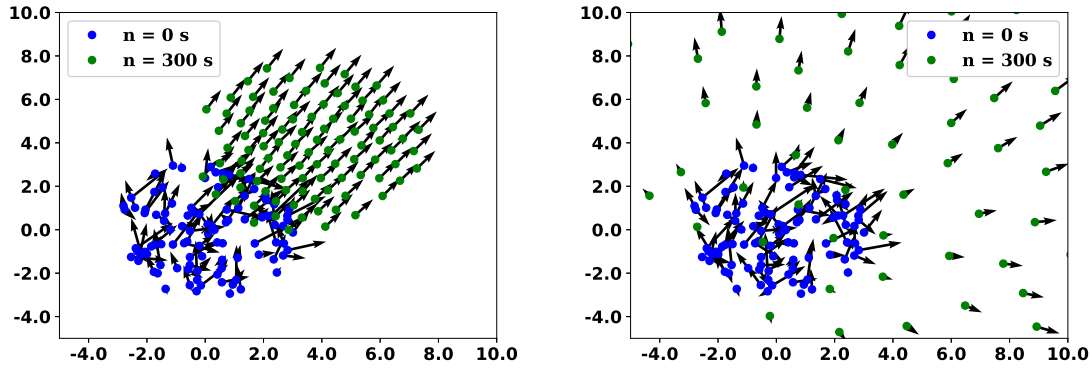
We report results comparing (3.3) and (3.2) for point masses with fully controllable accelerations. This simple setting allows for an exploration of the effect of different system parameters such as initial velocity or communication radius, to determine experimentally the scenarios on which the aggregation GNN offers good performance. We study the case of transfer learning, where we train the model in one network but test it in another (for example, with different number of agents), and also by exporting the trained architecture to other physical models beyond the point-mass model, as shown in the AirSim simulator.

Learning to flock with point masses. First, we compare the performance of the GNN controller with $K = 3$ to the local controller $\mathbf{u}_i^\dagger$ (3.3). Figure 3.1a depicts the magnitude of velocity differences between agents over the course of a trajectory in terms of the population mean and standard devision. The GNN converges much more rapidly and, unlike the local controller, approaches a perfect velocity consensus. Part of the reason for this is explained in Figure 3.1b, which plots agents' minimum distance to any neighbor over time. The GNN control approaches a uniform flock spacing, but the local controller fails to stop agents from dispersing quickly enough. Soon the local controller's network has become completely disconnected as agent distances exceed their communication range of $R = 1$. One flock trajectory is depicted for the GNN controller in Fig. 3.1c and



(a) Average difference in velocities

(b) Average minimum distance to a neighbor



(c) Flock positions using the GNN

(d) Flock positions using the local controller

Figure 3.1. The GNN ($K = 3$) maintains a cohesive flock, while the local controller allows the flock to scatter.

the local controller in Fig. 3.1d. Each diagram shows the initial agent positions and velocities at time $n = 0$ and then at $n = 300$, qualitatively illustrating the stable flocking of the GNN and failure of the local controller.

Next, we study the performance of the GNN controller (for different values of K) compared to both the local controller $\mathbf{u}_i^\dagger$ and the global controller $\mathbf{u}_i^*$ [eq. (3.2)] under different flocking scenarios; namely, different initial velocities (Fig. 3.2a), communication radius (Fig. 3.2b), and number of agents (Fig. 3.2c). Then, we also consider different aggregation GNN architecture hyperparameters (Fig. 3.2d. We recall that the performance is measured by the cost metric defined in (3.5). In general, we observe that the GNN cost is bounded by the global controller as a best-case baseline, and the the local controller as the worst-case baseline, as expected, showing, in many cases, a marked improvement over the local controller $\mathbf{u}_i^\dagger$.

More specifically, in Fig. 3.2a, we fix the communication radius $R = 1.0$ m for $N = 100$ agents. The $K = 1$ controller, which uses the same data as the local controller, performs an order of magnitude worse than

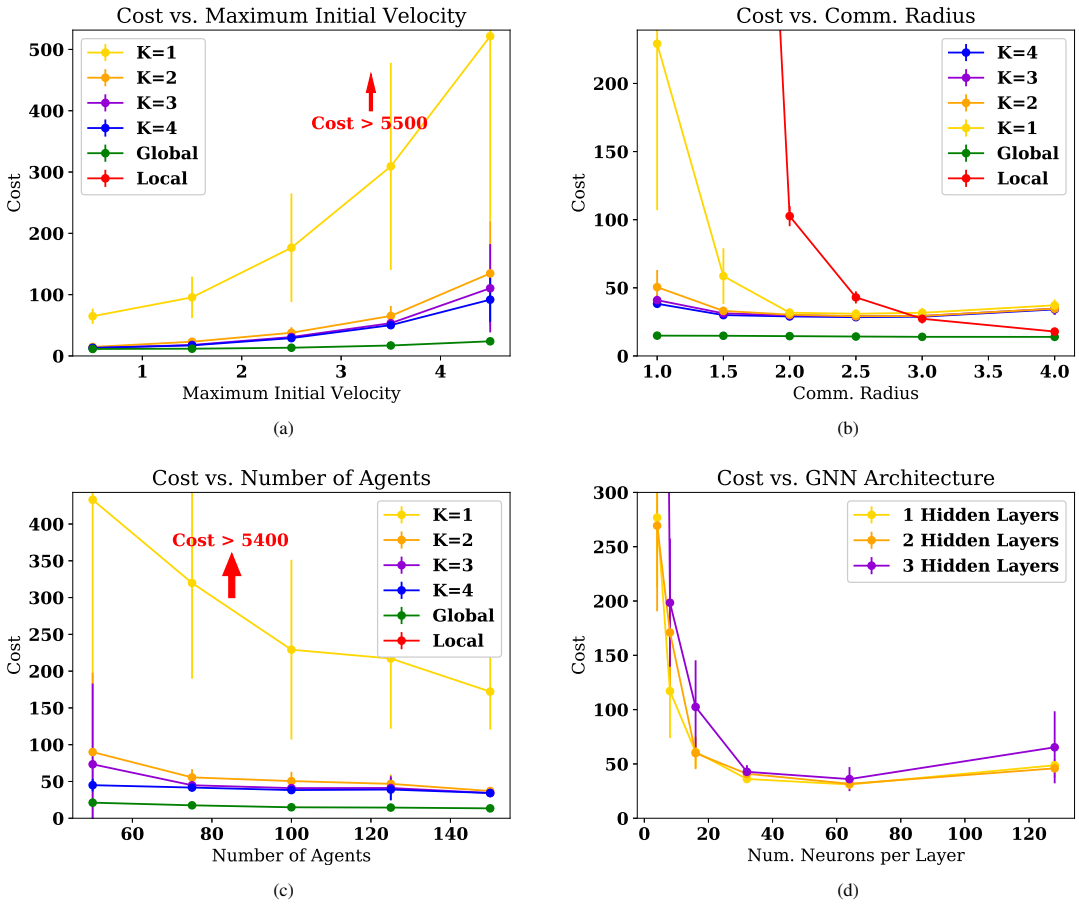


Figure 3.2. The flock's maximum initial velocities, communication radius, and the number of agents are key parameters affecting the control cost. The GNN architecture uses 2 hidden layers with 32 neurons each.

$K = 2$ to $K = 4$, whose performance is comparable. At the highest initial velocity of $v_{init} = 4.5$ m/s, the $K = 4$ GNN performs slightly better than the rest. Fig. 3.2b shows that the flocking problem becomes more challenging as the communication radius decreases, for fixed maximum initial velocities, $v_{init} = 3.0$ m/s, and $N = 100$ agents. The difference between the GNNs is most dramatic at $R = 1.0$ m, where the $K = 3$ and 4 controllers perform much better than all but the global controller. It is interesting to note that the cost of the local controller dips below the GNN controllers for large communication radii such as $R = 4.0$ m. This may be due to the lack of truncation of the GNN features in (3.4). In Fig. 3.2c, we observe that the flocking cost per agent, computed by (3.5), decreases in value and variance as the flock size increases, with fixed $R = 1.0$ m and $v_{max} = 3.0$ m/s. The GNN approach generalizes easily to 150 agents, with no penalties on larger

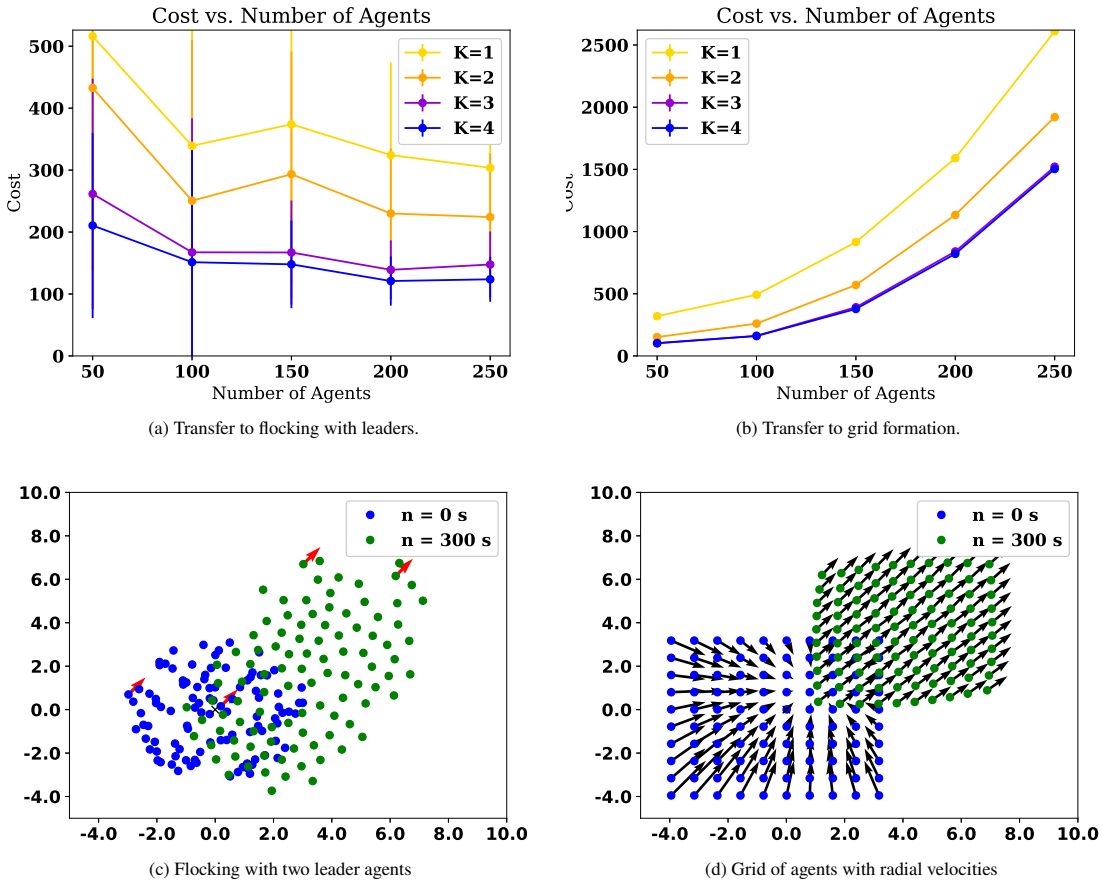
flocks. In Fig. 3.2d, we observe that the GNN architecture resulting in the lowest control cost uses 1-2 hidden layers and 32-64 neurons, for fixed $v_{init} = 3.0$, $N = 100$, $R = 1.0$ m. For smaller architectures, we observe under-fitting.

Transfer to Leader Following. Next, we investigate a new application of flocking in the presence of leader agents, with which the rest of the flock must align velocities. This application is extremely relevant for human-robot interaction for control of large swarms, and has not been previously explored by [52]. In the previous section, the flock positions and velocities are initialized at random. This induces a symmetry in the configuration - the distribution of velocities in an agent's 1-hop neighborhood is the same as in its 2-hop or 3-hop neighborhood. Now, our goal is to investigate applications in which this symmetry is not present, to emphasize the capabilities of the aggregation GNN for $K = 3$ and $K = 4$. All models in this experiment were trained for $v_{init} = 3.0$ m/s, $N = 100$, and $R = 1.0$ m.

We first examine the transfer of the trained controllers to a system with two leader agents that have equal velocities that remain constant throughout an episode. That is, we train the architecture on examples of the symmetric network, but we test it on the leader system. Results are shown in Fig. 3.3c. We can observe that the $K = 3$ and $K = 4$ aggregation GNNs provide the best performance which is in agreement with the intuition that 2-hop and 3-hop aggregations allow the agents to respond more quickly to a leader agent who may be multiple hops away. In the second scenario, agents are initialized in a grid, with velocities radially inwards toward the centroid of the flock. The initial velocities of agents are initialized to be proportional to their distance from the center of the flock. Results are shown in Fig. 3.3b. As the number of agents increases, we see the cost increases exponentially because the initial velocities increase, which is consistent with Fig. 3.2a. This scenario is particularly prone to the scattering behavior of Fig. 3.1d due to the high probability of near-collisions, but the $K = 3$ and $K = 4$ controllers successfully align the agents' velocities and promote regular spacing among agents.

C High order dynamics

The ideal point masses provide a convenient benchmark for testing the effects of system parameters, but ultimately, we are most interested in flocking for robotic systems. Our goal in this section is compare several approaches to flocking in large teams of up to 50 quadrotors in simulation. Testing in the AirSim simulator



(a) Transfer to flocking with leaders.

(b) Transfer to grid formation.

(c) Flocking with two leader agents

(d) Grid of agents with radial velocities

Figure 3.3. The trained GNN is transferred to new challenging scenarios with large numbers of agents.

allows us to test our controllers in the presence of higher order dynamics, slower control rates, and latency in observations [55].

The first challenge is that the AirSim simulator runs in real time, and produces an order of magnitude fewer training data points as compared to the point-mass simulation. To speed up training, we attempt to train on a simulated point mass system that was tuned to the parameters of AirSim. Since, we were not able to achieve an exact control frequency for the simulation, we instead observed that commands were issued at a variable time interval, with a mean of 0.12 seconds, and a standard deviation of 0.018 seconds. Therefore, we adapted the point mass simulation to match the AirSim simulation by sampling $T_s \sim \mathcal{N}(0.12, 3 \times 10^{-4})$ and applying the linear model in (1.1). The inputs and outputs of the controllers were scaled by a factor of $l = 6$ before evaluation of the GNN so that the optimal spacing dictated by the potential function (3.1) does not result in

collisions. The scaling of the control actions and features was performed by: $\mathbf{x}_{n+1} = f(\mathbf{x}_n, l \cdot \pi(\mathbf{x}_n/l))$, where f is the dynamics of the system, and $l = 6$ is the scaling factor.

The second challenge was converting from the desired acceleration values produced by the controller to the desired roll and pitch commands. We follow the approach of [56], which linearizes the dynamics of the robot about the hover point. In our case, the roll, ϕ, and pitch, θ, are proportional to the desired acceleration in the x and y dimensions. For a fixed yaw orientation, we used the approximation: $\phi = \mathbf{u}_2/g$ and $\theta = -\mathbf{u}_1/g$. The signs are inconsistent with those of [56] due to the AirSim convention that the $+Z$ dimension is down. This approximation is only valid for small accelerations, so the acceleration inputs were clipped to a range of $[-3.0, 3.0]$ m/s^2. Performance of the controllers was quantified using the variance of velocities in (3.5). We initialize the flock uniformly spaced in a grid formation with a distance of about 4.8 m between agents, and with x/y velocities sampled uniformly on the range $[-3.0, 3.0]$ m/s. The communication radius in AirSim was fixed at 9 m.

Fig. 3.4 compares the performance of controllers trained in simulation against those trained on the stochastic point mass model. The results of all eight GNN models are framed by the local and global controller as the upper and lower baselines. The GNN controller with $K = 4$ trained in AirSim outperformed all other learned controllers. For $K = 1$ and $K = 2$, the controllers trained in simulation and AirSim had similar performance, but as additional aggregation operations are added, the benefit of training in AirSim becomes more obvious. We believe that the $K = 4$ model trained on point masses has overfit to the ideal dynamics, so the performance degrades. Both in simulation and in point-mass experiments, we observed a failure mode, in which a small group of agents is moving too quickly and escapes from the rest of the group. This small sub-flock typically exhibits flocking behavior among the several agents, but has no ability to re-join the flock, because it is permanently outside of the communication range of the rest of the agents. This drawback results from the lack of hard constraints on the connectivity in the system.

D Summary

We have demonstrated the utility of aggregation graph neural networks as a tool for automatically learning distributed controllers for large teams of agents with coupled state dynamics and sparse communication links. We envision the use of an Aggregation GNN-based controller in large-scale drone teams deployed into

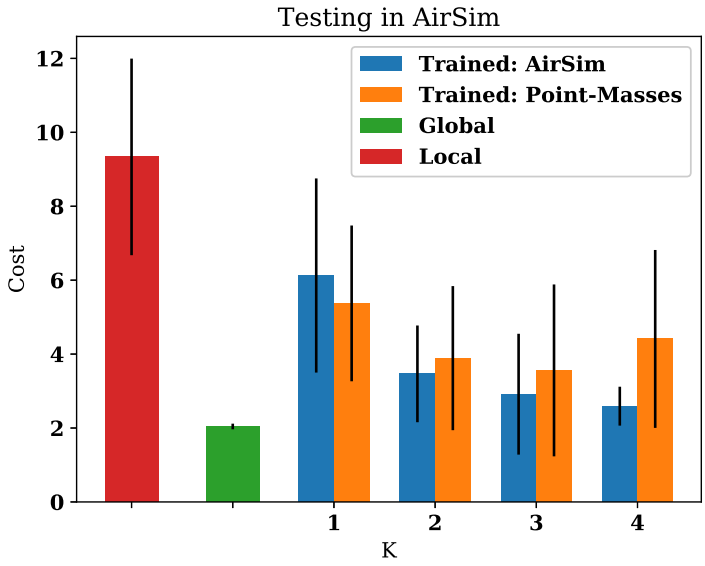


Figure 3.4. Four models were trained in AirSim and four on the stochastic point masses, and then all models were tested in AirSim. We visualize this experiment in a **video** provided along with this work.

communication-limited environments to accomplish coverage, surveillance or mapping tasks. In these settings, it is critical for agents to incorporate information from distant teammates in spite of local communication constraints; we show aggregations GNNs can be extended to accomplish this even with the time-varying agent states and time-varying communication networks typical of mobile robots. In experiments, learning decentralized controllers for flocking and additional applications confirms the value of multi-hop information to performance and robustness to number of agents and communication radius. In future work, enforcing state or input constraints could help avoid these failure modes.

Figure 3.5. Screenshot of drones in AirSim.

Chapter 4

Inverse Optimal Planning for Air Traffic Control

Air traffic controllers (ATC) must follow a complex set of regulations, including requirements on spacing between airplanes, weather restrictions, and airport-specific departure and arrival protocols. Additionally, experienced ATCs often formulate strategies that balance various demands that arise from the complex interplay between these factors. The demand for ATC services, which are already stretched thin, will further increase due to the rapid progress in the field of aerial robotics.

We tackle the problem of building an Autonomous ATC that could significantly reduce the load on the human operators. In particular, we envision a system that concisely describes the rules of air traffic control and supports dense autonomous air traffic around commercial airports. There are several key challenges in building such an autonomous system. First, there are various deterministic and stochastic variables, such as traffic density, weather, regulatory requirements, and local geography. Often, there are multiple compliant ways, that can be qualitatively very different, to route air traffic. While a human operator can seamlessly optimize across these factors, it is not clear how an algorithmic system should weigh and jointly optimize across all these different criteria to mimic the choices of the human ATC. Secondly, most of the existing ATC services are developed for standard fixed-wing aircraft and helicopters. It is not clear how the current ATC system should be extended to novel aerial vehicles, such as micro UAVs and VTOL vehicles, which might have completely different dynamic behavior. Finally, from the algorithmic point of view, such a planning task requires optimization across multiple dynamic agents, which quickly becomes intractable as the number of vehicles increases.

Our novel approach combines search-based motion planning and inverse reinforcement leaning to address these challenges. The key technical insight is that, given a planner, we can learn a reward function that an ATC might be optimizing by leveraging aircraft traces available via the U.S. Federal Aviation Administration's Aircraft Situation Display to Industry feed. In particular, we learn the parameters of the reward function that correspond to how different factors are considered for trajectory selection.

The resulting trajectories attempt to imitate real air-traffic and are shown to be safe and feasible. The learned cost functions are interpretable and can be compared to existing standard procedures. Additionally, the decoupling between the planner and the reward functions means that we can change the dynamics to those of a new aircraft or aerial robot and the system can adjust without retraining. Further, leveraging robotic path-planners also helps with computational challenges and eliminates the need for directly learning a control policy.

There are significant efforts to redesign the air traffic control system to enable autonomous decision making. One popular research direction is focused on conflict resolution to prevent vehicles from entering unsafe states. For example, [57] aims to quantify the complexity of the current air traffic situation and provides a scheme for distributed control that ensures safety and eliminates the possibility of collisions. [58] develops a Markov Decision Process-based system for resolving conflicts in flight plans. There has been a significant thrust in the development of air traffic simulators and interactive systems for experiments with control algorithms and human factors. [59] describes a simulator developed by NASA and an example of a reinforcement learning approach for an airplane's greedy optimization of arrival times. Rather than manually designing a reactive system that eliminates conflicts, we seek to learn how to generate feasible trajectories for open-loop control in dense air traffic.

In mobile robot navigation, the penalty on unsafe states can be formalized as a spatial potential field. The artificial potential field is a popular method for efficient obstacle avoidance behaviors [60]. First, a potential field is generated based on known obstacles, and then the gradient of this potential field determines the direction of motion. One major drawback is the lack of guarantees for arrival at the origin or obstacle avoidance. Getting stuck in local minima of the potential functions is a risk [60]. The evolutionary artificial potential function method has also been used for real-time robot path planning in the presence of dynamic obstacles [61].

Potential functions have been used extensively to model air traffic. [62] proposes a potential-field based

system that can encode the effects of factors such as weather and air traffic density to manage en-route traffic. [63] focuses on the problem of re-routing of air traffic due to weather by generating potential functions and addresses the problems of local minima.

These works use heuristics to avoid local minima in the potential field. They also neglect the problem of path planning in dense airspace around airports immediately prior to landing. We follow the approach of [64], which uses an artificial potential function as a penalty so that an optimal planner avoids potentially unsafe states. We extend this approach to the Dubins Airplane model and learn the potential function.

Inverse reinforcement learning is the problem of using expert demonstrations to learn the reward function that an expert is maximizing. This reward function can then be used to determine a controller that imitates the trajectories of the expert. Approaches such as behavior cloning [65] seek to directly find a mapping from states to experts' actions, but this may generalize poorly to new situations. Other approaches include maximum margin planning [66] and feature expectation matching [67], but these suffer from an ambiguity because one policy can be optimal for many different reward functions. We apply the maximum entropy inverse reinforcement learning algorithm described by [68]. This approach has been extended to deep reward functions and policies [69] and has been used in conjunction with planning for manipulation tasks [70].

This chapter has been published as [71]. An open-source implementation of our work can be found at `https://github.com/katetolstaya/flight_data`. A video demonstration of the approach is available at `https://youtu.be/5HasgHNl-XY`.

A Planning using the Dubins Airplane Model

We model the state of a single airplane using the Dubins airplane model, a four-dimensional system with a configuration space, $\mathcal{S} = \mathbb{R}^3 \times [-\pi, \pi]$, with $\mathbf{s} = (x, y, z, \phi)$, where x, y and z describe the coordinates of the airplane in three-dimensional Euclidean space, and $\phi \in [-\pi, \pi)$ is the bearing of the airplane relative to the $+x$ axis [72]. This airplane model assumes a fixed speed of v in the xy-plane.

The system is controlled through the first derivatives of altitude and bearing, $\dot{\phi}$ and $\dot{z}$, with control inputs denoted as u_z and u_ϕ, respectively. The system can be described as:

$$\dot{\mathbf{s}} = [\dot{x}, \quad \dot{y}, \quad \dot{z}, \quad \dot{\phi}]^T = [v\cos\phi, \quad v\sin\phi, \quad u_z, \quad u_\phi]^T \tag{4.1}$$

This model can be used to plan a trajectory from an initial state $\mathbf{s}_0 = (x_0, y_0, z_0, \phi_0)$ to a given a goal configuration $\mathbf{s}_g = (x_g, y_g, z_g, \phi_g)$. This information is given to each airplane by air traffic control when the airplane approaches an airport. For example, $\mathbf{s}_g$ may be the location and orientation of an assigned runway. We assume that these assignments are provided prior to planning. The problem of planning a safe, minimum-length trajectory from the current state of the airplane to the goal can be formalized as follows:

$$\begin{aligned} &\underset{\mathbf{s}(t)}{\operatorname{argmin}} \int_{t_0}^{T} \|\dot{\mathbf{s}}(t)\|\, dt \\ &\quad \text{s.t. } \dot{\mathbf{s}}(t) = \big(\cos\phi, \sin\phi, u_z(t), u_\phi(t)\big) \\ &\qquad \mathbf{s}(t_0) = \mathbf{s}_0, \mathbf{s}(T) = \mathbf{s}_g \\ &\qquad \mathbf{s}(t) \in \mathcal{S}^{safe}, \mathbf{u}(t) \in \mathcal{U} \end{aligned} \tag{4.2}$$

where $\mathcal{S}^{safe}$ is the set of safe states and $\mathcal{U}$ is the set of allowed control inputs. To enable learning the set of safe states $\mathcal{S}^{safe}$, we can reformulate (4.2) to use a soft penalty on possibly unsafe states, following the approach of [64], which uses hard constraints in addition to a potential function that guides the planned trajectory farther away from obstacles. The penalty term, $J\big(\mathbf{s}(t)\big)$, will be learned from data. For now, we express the new minimum path length planning problem as:

$$\begin{aligned} &\underset{\mathbf{s}(t)}{\operatorname{argmin}} \int_{t_0}^{T} \Big(1 + J\big(\mathbf{s}(t)\big)\Big) \|\dot{\mathbf{s}}(t)\|\, dt \\ &\quad \text{s.t. } \dot{\mathbf{s}}(t) = \big(\cos\phi, \sin\phi, u_z(t), u_\phi(t)\big) \\ &\qquad \mathbf{s}(t_0) = \mathbf{s}_0, \mathbf{s}(T) = \mathbf{s}_g \\ &\qquad \mathbf{u}(t) \in \mathcal{U} \end{aligned} \tag{4.3}$$

The motion cost of a trajectory, $\tau : [0, \tau] \to \mathcal{S}$, is the sum of the path length and the line integral of the penalty:

$$C(\tau) = \int_{t_0}^{T} \Big(1 + J\big(\mathbf{s}(t)\big)\Big) \|\dot{\mathbf{s}}(t)\|\, dt. \tag{4.4}$$

Solving the motion planning problem (4.3) requires searching the space of all feasible trajectories. The continuous state and time problem is intractable, so we follow the approach of [73] by discretizing the system in time with an interval of $\Delta t = 30$ seconds. We also discretize the controls to obtain a set of fixed-time

motion primitives. The bearing is chosen from the set $u_\phi \in \{-\Delta\phi, 0, \Delta\phi\}$ and the altitude change is chosen from the set $u_z \in \{-\Delta z, 0, \Delta z\}$. The motion primitives induce a discretization of states and enable the use of search-based motion methods for planning feasible and resolution-complete solutions [73]. While continuous states are denoted $\mathbf{s} \in \mathcal{S}$, we denote the discretized states as $\bar{\mathbf{s}} \in \mathcal{G}$, where $\mathcal{G}$ is a 4-dimensional grid, $\mathcal{G} \subset \mathcal{S}$. The induced discretization has a resolution of $\boldsymbol{\rho} = [\rho_x, \rho_y, \rho_z, \rho_\phi]$, so the conversion from $\mathbf{s}$ to $\bar{\mathbf{s}}$ can be expressed using the floor function $\lfloor \, \rfloor$:

$$\bar{\mathbf{s}} = \left[\left\lfloor \frac{\mathbf{s}_x}{\boldsymbol{\rho}_x} \right\rfloor, \left\lfloor \frac{\mathbf{s}_y}{\boldsymbol{\rho}_y} \right\rfloor, \left\lfloor \frac{\mathbf{s}_z}{\boldsymbol{\rho}_z} \right\rfloor, \left\lfloor \frac{\mathbf{s}_\phi}{\boldsymbol{\rho}_\phi} \right\rfloor \right] \tag{4.5}$$

To solve the discretized problem, we apply the Anytime Repairing A* (ARA*) algorithm for finite-time path planning [74]. Please note the ϵ-suboptimality guarantee for the ARA* algorithm, which is expressed as:

$$C(\tau^*) \leq C(\tau) \leq \epsilon C(\tau^*) \tag{4.6}$$

where $\epsilon \geq 1$ and τ^* is the optimal path and $C(\tau^*)$ is its cost. The ability to sample suboptimal trajectories allows us to visit and learn about a larger variety of states during inverse reinforcement learning. Given the continuous trajectory defined by a series control inputs, we then use trajectory refinement to produce smoother trajectories using spline interpolation [73].

To use the ARA* algorithm for motion planning for fixed-wing vehicles, we must use a heuristic that provides a lower bound on the distance to the goal. For the non-holonomic Dubins airplane model, there are existing methods for computing the optimal path length from a start state to a goal state, but they are expensive to compute and require the consideration of low-altitude and high-altitude cases [72]. To simplify this computation, we always assume the high altitude case, which allows adding a helical descent or ascent without worrying about collisions with the ground. Our heuristic may therefore be inadmissible in difficult low-altitude cases, but is much more computationally efficient. The computation of this heuristic is summarized in Algorithm 2: First, we use the Dubins car model to compute the minimum path length from the start state to the goal state in the xy plane. Following the approach of [75], to find the shortest length path, we check each of six types of Dubins Car paths, with each path consisting of three segments: Right (R), Straight (S) and Left (L) [75]. Then, we compute the minimum time necessary for the ascent or descent. Then, if there is not enough time for the airplane to change altitude, we add helical sections to the trajectory to

allow the airplane to descend in the z dimension and return to the same position and bearing in the xy plane.

Algorithm 2 Dubins Airplane Heuristic

Input: Start $\mathbf{s}_0$, goal $\mathbf{s}_g$, forward speed v, max rate of climb Δz, turning rate $\Delta\phi$
Output: Minimum path length from start to goal, d_{min}

1: Compute Dubins car distance d_{xy} using the LSL, RSR, LSR, RSL, RLR, LRL paths with curvature $\kappa = \Delta\phi/v$
2: Compute the minimum time from start to goal in the xy plane

$$t_{min} = d_{xy}/v$$

3: Compute the minimum time for ascent/descent

$$t_z = |z_g - z_0|/\Delta z$$

4: **while** $t_z > t_{min}$ **do**
5: Add a helical ascent/descent to the trajectory

$$t_{min} = t_{min} + \frac{2\pi}{\Delta\phi}$$

6: **end while**
7: Compute the minimum path length from start to goal:

$$d_{min} = \sqrt{(v \cdot t_{min})^2 + (z_g - z_0)^2}$$

B Learning From Demonstrations

We assume that the set of safe states $\mathcal{S}$ in (4.2) is unknown. These conditions may be determined by the layout of the airspace at an airport, the motion of other airplanes, or weather. Our goal in this work is to learn about the unsafe conditions through inverse optimal control. To enable learning J using gradient descent, we use the soft penalty formulation in (4.3).

In order to learn the true penalty $J(\mathbf{s})$, we require a data set of expert demonstrations, $\mathcal{D} = \left\{s_i^e(t)\right\}_{i=1,\ldots,M}$. Although we do not know the true penalty $J(\mathbf{s})$, we have a current best estimate, $J_\theta(\mathbf{s})$. This cost, J_θ, is a function of non-linear features $\mathbf{f}$ of the state $\mathbf{s}$, defined in Section C along with particular examples of cost functions. We parametrize J_θ as linear in the features of the state $\mathbf{f}(\mathbf{s})$:

$$J_\theta(\mathbf{s}) = \theta^T \mathbf{f}(\mathbf{s}). \tag{4.7}$$

The motion planner can use this estimated J_θ to plan trajectories satisfying (4.3). Our next goal is to formulate the loss function that will allow us to update J_θ using trajectories from the expert and learner.

We assume that the demonstrated expert trajectories follow the principal of maximum entropy [76], which states that the probability of an expert's trajectory τ with a lower cost is exponentially more likely to be selected than a trajectory with a higher cost:

$$\mathbb{P}(\tau) \propto e^{-C(\tau)} \tag{4.8}$$

Using this assumption, we can apply the maximum entropy inverse reinforcement learning algorithm described by [68], and its deep learning counterpart [69]. A detailed comparison of related works can be found in [70]. This approach seeks to find the cost function J_θ that *maximizes* the log likelihood of expert trajectories $\mathcal{D}$:

$$\mathcal{L}(\theta) = \log \mathbb{P}(\mathcal{D}, \theta | J_\theta) \tag{4.9}$$

If the cost function J_θ is a linear function the features, this problem is convex. [68] shows that the gradient of this objective is the difference in feature counts along the trajectories of the expert and the learner. To compute the gradient of $\mathcal{L}$ with respect to θ, we need to first introduce the state visitation counts of the expert computed using the data set $\mathcal{D}$:

$$\mathbf{f}_\mathcal{D} = \sum_{\tau \in \mathcal{D}} \sum_{\mathbf{s} \in \tau} \mathbb{P}(\tau) \mathbf{f}(\mathbf{s}) \tag{4.10}$$

In [68], the expert's empirical feature counts are compared to the expectation of the learner's state visitation counts:

$$\mathbb{E}[\mathbf{f}_\theta] = \sum_{\mathbf{s} \in \mathcal{S}} \mathbf{f}(\mathbf{s}) \mathbb{P}(\mathbf{s} | J_\theta) \tag{4.11}$$

Using these two quantities, we can express the gradient of $\mathcal{L}$ to be equal to the difference in feature counts along the trajectories of the expert and the learner [68, 69]:

$$\nabla_\theta \mathcal{L} = \mathbb{E}[\mathbf{f}_\theta] - \mathbf{f}_\mathcal{D} \tag{4.12}$$

As in [77], we assume that the distribution of the learner's sampled trajectories is uniform. Therefore, the expectation $\mathbb{E}[\mathbf{f}_\theta]$ can be estimated using samples from learner with the current cost function J_θ and we can use

Algorithm 3 Inverse Optimal Control for Air Traffic

repeat
  Receive expert trajectories ordered by arrival time

$$\mathcal{D} = \left\{\mathbf{s}_i^e(t)\right\}_{i=1,\ldots,M}$$

  for $i = 0, 1, \ldots, M$ **do**
    Obtain start state, $\mathbf{s}_0$, and end state, $\mathbf{s}_g$, of $\mathbf{s}_i^e(t)$
    Obtain other airplanes' trajectories, $\mathbf{s}_k^o(t)$ for $k < i$
    Use ARA* planner to minimize J_θ over $\mathbf{s}(t)$
    Compute stochastic gradient:

$$\hat{\nabla}_\theta \mathcal{L} = \sum_{t=t_0}^{T} f\left(\mathbf{s}(t)\right) - \sum_{t=t_0}^{T_i} f\left(\mathbf{s}_i^e(t)\right)$$

    Update cost parameters θ with step size α:

$$\theta_{t+1} = \theta_t + \alpha \hat{\nabla}_\theta \mathcal{L}$$

  end for
until convergence

the stochastic gradient computed using trajectories from the learner with the current cost J_θ, $\left\{\mathbf{s}_i(t)\right\}_{i=1,\ldots,N}$, and trajectories from the expert, $\left\{\mathbf{s}_i^e(t)\right\}_{i=1,\ldots,M}$:

$$\hat{\nabla}_\theta \mathcal{L} = \underbrace{\frac{1}{N}\sum_{i=1}^{N}\sum_{t=t_0}^{T_i} f\left(\mathbf{s}_i(t)\right)}_{\text{learner}} - \underbrace{\frac{1}{M}\sum_{i=1}^{M}\sum_{t=t_0}^{T_i} f\left(\mathbf{s}_i^e(t)\right)}_{\text{expert}} \tag{4.13}$$

Gradient ascent on the objective $\mathcal{L}(J)$ will increase the cost of the states that the learner visits, but the expert does not, so that the learner will avoid those states in the future. In practice, we set $M = N = 1$.

We summarize our approach in Algorithm 3. The learner obtains sets of time-synchronized expert trajectories for the landings of two or more airplanes. Then, for each trajectory in the expert's data set, the ARA* planner is used to plan a trajectory between the same start and end states. Given the expert's and learner's trajectories, we can then compute the stochastic gradient and perform a gradient step on the θ parameter of the cost function. If the planner fails to produce a solution within the time limit, we use only the expert's trajectory for the gradient computation.

C Safety in a Multi-agent System

Our next goal is to use prior knowledge to add structure to the cost function J to speed up learning. We assume that the centralized air traffic controller plans the landing trajectories for airplanes in the order of their arrival. When planning the trajectory for every new airplane, $\mathbf{s}(t)$, the trajectories of the n previous airplanes are known, denoted $\left\{\mathbf{s}_k^o(t)\right\}_{k=1,\dots,n}$. We also know the location of the destination airport, $\mathbf{s}_a$.

Also, recall that the planning problem is computed in a discrete state space $\bar{\mathbf{s}} \in \mathcal{G}$, so we will only need to compute the cost of states on this discrete grid and can use the conversion from $\mathbf{s}$ to $\bar{\mathbf{s}}$ in (4.5). Therefore, rather than directly learning J, we only need to learn $\bar{J} : \mathcal{G} \to \mathbb{R}$. To relate this form to the previous notation, this is equivalent to choosing a feature extraction function $\mathbf{f}(s)$ that converts the continuous states $\mathbf{s}$ to discrete states $\bar{\mathbf{s}}$ and then applies additional non-linear operations.

We assume that $\bar{J}$ depends on two types of safety constraints: location relative to a specific airport, and the pairwise spacing between airplanes. The first component of the cost function $\bar{J}$ controls the airspace around airport a and can be expressed as $\bar{J}_a\big(\bar{\mathbf{s}}(t)\big)$. The second cost function controls the pairwise spacing of airplanes and can be written as $\bar{J}_o\big(\bar{\mathbf{s}}(t), \bar{\mathbf{s}}_k^o(t)\big)$, where $\bar{\mathbf{s}}_k^o(t)$ is the location of a nearby airplane at time t. If there are n other airplanes in the area, we must sum this objective for all other airplanes: $\sum_{k=1}^{n} \bar{J}_o\big(\bar{\mathbf{s}}(t), \bar{\mathbf{s}}_k^o(t)\big)$. Therefore, $\bar{J}\big(\bar{\mathbf{s}}(t)\big)$ can be written as a sum of the two types of soft penalties:

$$\bar{J}(\mathbf{s}(t)) = \bar{J}_a\big(\bar{\mathbf{s}}(t)\big) + \sum_{k=1}^{n} \bar{J}_o\big(\bar{\mathbf{s}}(t), \bar{\mathbf{s}}_k^o(t)\big) \tag{4.14}$$

Next, we parametrize $\bar{J}_a$ and $\bar{J}_o$ to allow us to learn these functions using gradient descent. Motivated by planning in the presence of potential functions [64], we choose to represent $\bar{J}_a$ as a spatial potential field, approximated using a fine discretization of the input space. This approximation represents the cost function as a value for each state $\bar{\mathbf{s}} \in \mathcal{G}$ on a discrete grid. A cost function used for motion planning must be non-negative. Also, the formulation must be piece-wise linear in features of state, $\mathbf{f}(\mathbf{s})$, to satisfy the assumptions of [68]. Therefore, we use a cost function of the following form:

$$\bar{J}_a\big(\bar{\mathbf{s}}(t)\big) = \sum_{\bar{\mathbf{s}}_i \in \mathcal{G}} \max\left\{w_i, 0\right\} \mathbb{1}_{\bar{\mathbf{s}}(t)=\bar{\mathbf{s}}_i}\big(\bar{\mathbf{s}}(t)\big), \tag{4.15}$$

where $\mathcal{G} \subset \mathbb{R}^4$ is a finite set of all states within the controlled airspace and $\mathbb{1}$ is the indicator function. By

learning w_i for each discrete state in the space, we construct a look-up table of costs to enable efficient motion planning.

Next, we describe the penalty on pairwise distances between airplanes to control the airspace around each airplane. Each prior arrival is treated as a moving obstacle with a known trajectory, $\bar{\mathbf{s}}_k^o(t)$, and a cylindrical shape. We choose the penalty on getting too close to another airplane to be a linear drop-off potential function [78]:

$$\bar{J}_o\left(\bar{\mathbf{s}}(t), \bar{\mathbf{s}}_k^o(t)\right) = \mu \max\left\{\lambda_z - |\Delta_z(t)|, 0\right\} \max\left\{\lambda_{xy} - \sqrt{\Delta_x^2(t) + \Delta_y^2(t)}, 0\right\}, \tag{4.16}$$

where $\Delta(t) = \bar{\mathbf{s}}(t) - \bar{\mathbf{s}}_k^o(t)$ and $\Delta_x(t)$, $\Delta_y(t)$ and $\Delta_z(t)$ are the components of the difference in the discrete positions of the two airplanes at time t. We consider the relative positions in the horizontal and vertical dimensions separately since the clearance requirements in altitude may be different. The λ_{xy} and λ_z thresholds are learned through gradient descent, while the scaling factor μ is determined as a hyper parameter. The parameters for $\bar{J}_a$ and $\bar{J}_o$ can now be learned through stochastic gradient descent:

$$\theta = \left[\{w_i\}_{\bar{\mathbf{s}}_i \in \mathcal{G}}, \lambda_{xy}, \lambda_z\right] \tag{4.17}$$

The number of parameters w_i is large and equal to $|\mathcal{G}|$. In the next section, we describe the practical considerations of learning this set of parameters.

D Methods and System Architecture

Now, we describe the implementation of the inverse optimal control system that learns from the air traffic data set.

Dataset Processing. In this work, we narrow our focus to airplane landings at the Seattle-Tacoma airport (SEA), but this approach can be generalized to take-offs or inter-airport routing. Specifically, our goal is to mimic the Standard Terminal Arrival Routes at the SEA airport [79]. We use recorded trajectories from the 11^{th} - 13^{th} of January, 2016 as provided by FlightAware [1]. A visualization of current data is displayed in Figure 4.1. The data set provides the location of airplanes asynchronously at a time interval of about 30 seconds.

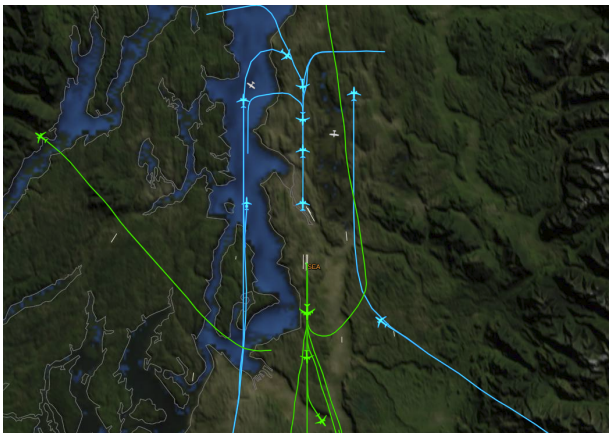


Figure 4.1. A visualization of the Seattle-Tacoma airport data provided by FlightAware, with arrivals indicated in teal and departures in green [1].

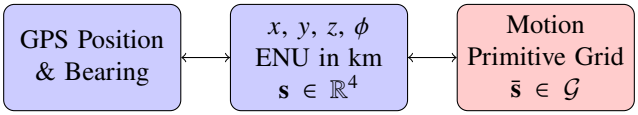


Figure 4.2. Three parametrizations are used for expressing airplane states. The raw data provided by FlightAware is given in GPS latitude, longitude and altitude. We can convert from GPS to a local east-north-up (ENU) Cartesian frame. Then, we discretize the state with a resolution of ρ to obtain the motion primitive grid coordinates via (4.5). Continuous state representations are denoted in blue and discrete - in red.

The airplane locations were provided as GPS measurements of latitude, longitude and altitude from onboard instrumentation. Bearing was estimated from subsequent observed locations. The provided WGS-84 measurement (GPS latitude, longtiude and altitude location) were converted to a local east-north-up (ENU) Cartesian system centered on the location of the Seattle-Tacoma airport [80, 81]. All distances are provided in kilometers and bearing angles in radians from the $+x$ axis. The $+z$ axis indicates up and is perpendicular to the tangent plane.

By utilizing the Dubins airplane model, we make a constant velocity assumption. Real airplane trajectories accelerate and decelerate, especially during take-offs and landings. Extending our approach to acceleration or jerk-based motion primitives as in [82] is left to future work.

Interpolation. The FlightAware data set provides waypoints at a time interval of about 30 seconds. We needed to perform interpolation, which is required for three reasons: 1) to generate time-synchronized trajectories for multiple airplanes, 2) to approximate the cost along a continuous trajectory, and 3) to generate dense

trajectory data for SGD updates. The interpolation is computed for the x, y and z components of the trajectory separately using the Univariate Spline module of the SciPy library [83]. The bearing of the aircraft is then computed using subsequent x_t, y_t, z_t locations, $\phi_t = \arctan2(y_{t+1} - y_t, x_{t+1} - x_t)$.

Planning. The ARA* planner is given a fixed time limit of 30 seconds. The planner does not always find a solution within the time limit, in which case we assume that the learner obtained a trajectory with zero cost and no states. This provides a lower bound on the learner's cost and assumes that the learner is able to instantly move directly from that start to the goal. In this case, we can use only the expert's trajectory for the stochastic gradient to update the cost function. The parameters of the motion primitives, such as the maximum turning rate and maximum rate of climb, were chosen based capabilities of commercial aircraft and tuned through a grid search procedure by comparing with actual trajectories. The parameters were chosen to be: forward velocity $v = 100$ m/s, rate of climb $\Delta z = 6$ m/s, time discretization $\Delta t = 30$ s, angular velocity $\Delta\theta_1 = 0.025$ radians/s. An additional allowed turning rate of $\Delta\theta_2 = 0.0025$ radians/s mitigated some of the imprecision resulting from the coarse time discretization. The Dubins airplane heuristic still uses the largest turning speed to compute an approximate lower bound on the path length. The airplane states were discretized to a resolution of $\rho_x = \rho_y = 125$ m in the x and y dimensions and $\rho_z = 50$ m in the z dimension. The bearing was discretized to $\rho_\phi = 0.05$ radians. Also, due to the discretization of the control space, it is extremely unlikely that the planned trajectories end up at exactly the goal state. Therefore, rather than having a single goal state, we use a goal region of $\pm[500, 500, 25]$ meters and ± 0.125 radians centered around the original goal state.

Learning. To fit the cost function $\bar{J}_a$ as described in (4.15), we need to store cost function values for each discrete state in the state space. We observe that the true structure of the data is sparse, so rather than storing a dense look-up table of cost values, we implement a sparse hash-map approach, which allows a constant-time evaluation of a state's cost. $\bar{J}_a$ is initialized to a cost of $w_i = 100$ for all states. When we observe a particular state and perform a gradient step, we store the new value in the hash table. To further improve computational efficiency, the cost function was discretized on a slightly coarser grid that that of the motion primitive discretization, with resolution 0.25 in the x and y dimensions and 0.125 in the z dimension. The bearing was discretized to 0.125 radians. A step size of $\alpha = 10.0$ was used for learning.

The discretization of the cost function presents a challenge if the localization information is not exact. The provided dataset uses GPS locations, which are susceptible to drifts and sensor noise. One possible

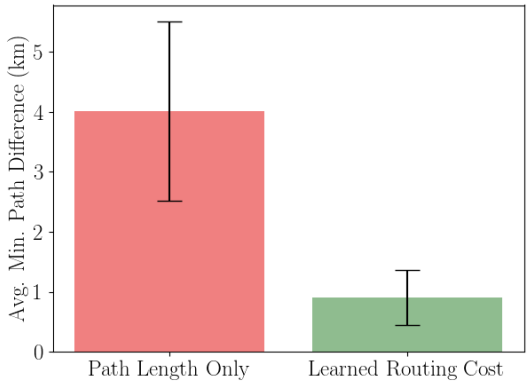


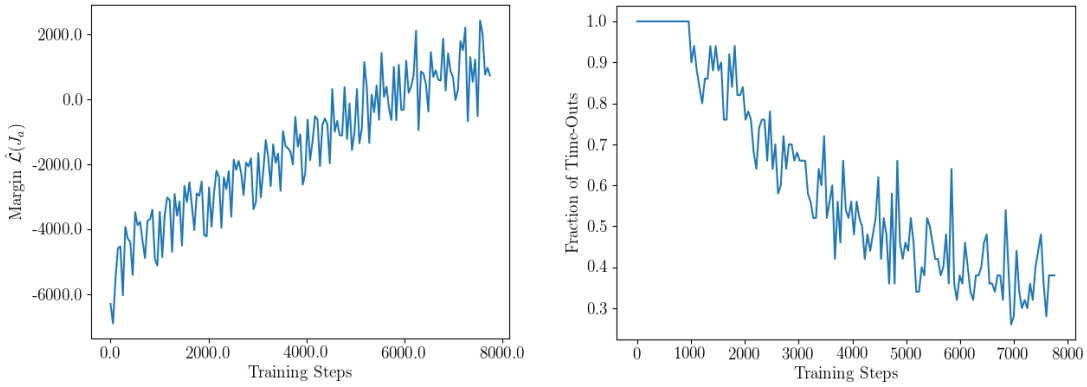
Figure 4.3. We quantify the planning performance of a planner that uses the learned $\bar{J}_a$ function and a planner that minimizes the path length criterion only. The average minimum path difference is $\frac{1}{N}\sum_{i=1}^{N}\frac{1}{T}\sum_{t=1}^{T}\min_u ||\bar{\mathbf{s}}_i(t)-\bar{\mathbf{s}}_i^e(u)||_2$, a measure of how far the learner's trajectory strays from the expert's demonstration on average. Error bars denote a one standard deviation bound and $N = 300$.

solution is evaluating the cost function at planning time by computing the average of multiple nearby cells of the cost function grid. This is expensive due to the sparse data structure used to store the values of the cost function. Instead, we add Gaussian noise to the state $\mathbf{s}_t$ before the gradient update of the cost function (4.13). A small amount of white Gaussian noise is added to the input state $\mathbf{s}_t$ before discretization: $\mathbf{w}_t \sim \mathcal{N}(0, \Sigma)$, where $\Sigma = [0.25, 0.25, 0.125]I$. The cost function $\bar{J}_o$ was initialized with overly conservative thresholds of $[\lambda_{xy} = 60, \lambda_z = 60]$ in discretized units, which is equivalent to $[7.5, 3]$ km. A step size of $\alpha = 0.01$ was used, with gradients larger than 100.0 clipped. The slope of the cost function is $\mu = 1.0$.

E Results

The arrival route function $\bar{J}_a$ has an extremely large number of parameters and therefore requires more data to train. We trained this function first without considering distances between airplanes. Then, we fixed the routing cost function while learning the inter-airplane safety function $\bar{J}_o$.

Learning the Airport Routing Cost. First, we train the airport routing cost, $\bar{J}_a$, that defines the allowed paths for airplanes arriving to the Seattle-Tacoma airport. Rather than the expensive computation of $\mathcal{L}(J_\theta)$ (4.9), we benchmark performance by the margin between the cost of the learner's trajectory and that of the



(a) Cost Margin $\hat{\mathcal{L}}(\bar{J}_a)$

(b) Fraction of Time-outs

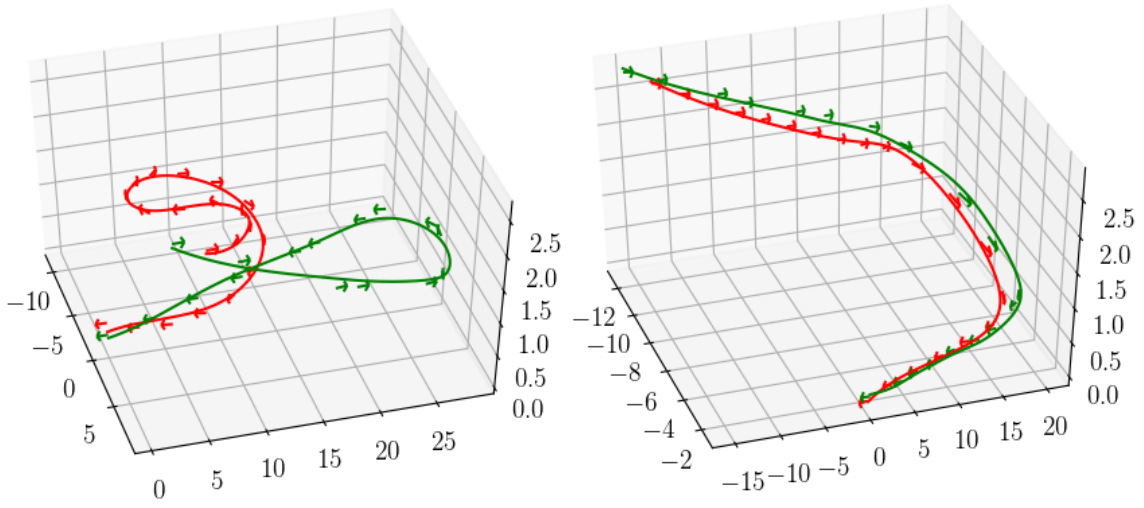
Figure 4.4. First, we train the airport routing cost function $\bar{J}_a$. In Fig 4.4a, we observe that, $\hat{\mathcal{L}}(\bar{J}_a)$, the margin between the expert's cost and the learner's cost increases during training, which we use as a proxy for the true optimization objective $\mathcal{L}(\bar{J}_a)$. The ARA* planner is given a fixed time budget and sometimes exceeds that time budget without finding a solution. Fig 4.4b shows the fraction of time-outs, which decreases during training. In both plots, each data point is the average of 50 consecutive training steps.

expert, given an expert trajectory $\bar{\mathbf{s}}_i^e(t)$ and a corresponding planned trajectory $\bar{\mathbf{s}}(t)$:

$$\hat{\mathcal{L}}(\bar{J}_a) = \sum_{t=t_0}^{T_i} \bar{J}_a\big(\bar{\mathbf{s}}(t)\big) - \sum_{t=t_0}^{T_i} \bar{J}_a\big(\bar{\mathbf{s}}_i^e(t)\big) \tag{4.18}$$

As the cost of the expert trajectory decreases relative to that of the expert, the objective increases and the probability of the expert's trajectory (4.9) increases. In Figure 4.4a, we observe that the benchmark $\hat{\mathcal{L}}(J_\theta)$ increases during training.

It is important to note that the use of a fixed time limit for the ARA* planner often produces time-outs. In this case, the gradient step is computed using only the expert trajectory, treating the learner's trajectory as a cost of zero. Also, to speed up learning for the first 1000 iterations, we used only the expert's trajectory for the gradient step since the fraction of time-outs is large at the beginning of training. It is interesting that the number of time-outs decreases during training, as we can see in Figure 4.4b. As the cost of the expert trajectories and the corresponding states decreases, the Dubins airplane heuristic becomes a tighter lower bound on the motion cost of the trajectories that move through those states. In Figure 4.5, we can see two examples of planned trajectories. We compare a trajectory planned using only the minimum path length objective to a trajectory planend using the learned $\bar{J}_a$ cost function. After training, the learner closely



(a) Minimum path length

(b) Using $\bar{J}_a$

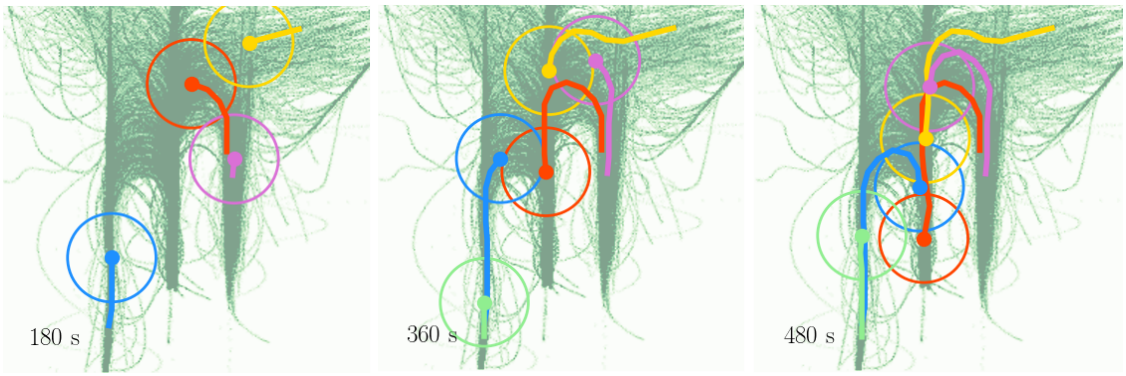
Figure 4.5. A comparison of the expert's and learner's 3D trajectories for two different objective functions. The actual ATC trajectory is denoted with green arrows and a green spline. The planner's path is marked with red arrows and interpolated by the red curve. The units are in kilometers in the ENU coordinate system. In 4.5a, the planner minimizes the path length alone, and in Figure 4.5b, the planner uses the learned airport routing cost, $\bar{J}_a$.

follows the expert's trajectory, with small oscillations in the z-axis. Figure 4.3 quantifies this comparison of the minimum path length planner and the routing cost planner over 1000 trajectories.

Learning the Inter-Airplane Safety Cost. Now, we turn to learning the cost function that controls the spacing between pairs of airplanes, $\bar{J}_o$. We examine an example of five concurrent landing trajectories planned using the learned costs $\bar{J}_a$ and $\bar{J}_o$ in Figure 4.6. The cost function $\bar{J}_a$ is visualized in green, with darker values indicating lower cost, and lighter values indicating higher cost. The five airplane trajectories are indicated with colored curves, with the current location of each airplane marked with a small filled circle. The thresholds of the $\bar{J}_a$ function are marked with large unfilled circles of corresponding colors. If a sixth airplane approached the airport, it would consider these unfilled circles to be states with very high cost and would avoid these regions, eliminating the posibility of collisions.

F Summary

This work develops the method for search-based planning using cost functions learned through inverse optimal control. We demonstrate that the learned cost functions encode the implicit safety criteria of human ATC



(a) t = 180 s (b) t = 360 s (c) t = 480 s

Figure 4.6. A top down view of five trajectories produced by the learner using both $\bar{J}_a$ and $\bar{J}_o$, with the trajectories of the airplanes denoted in red, blue, yellow, purple and green, and their current locations denoted with a small filled circle. The large unfilled circles denote the threshold of the $\bar{J}_o$ cost, preventing airplanes from getting too close to each other. The learned cost function $\bar{J}_a$ is in the background indicated by green shading, with dark green indicating low cost states, and white indicating high cost states. We observe that the controller maximizes the efficiency of this landing sequence by pushing the trajectories of the airplanes as close together as allowed by the safety criterion.

trajectories while maintaining high efficiency comparable to that of human operators. Our approach is well suited for planning trajectories over longer distances of tens of kilometers, but the Dubins airplane dynamics model has obvious limitations due to the coarse discretization of the control space, which are especially apparent during the last stages of a landing. Given a data set with more precise location measurements at a higher frequency, one possible solution could be to directly learn the motion primitives from data. This approach would still need to be verified in a high-fidelity air traffic simulation.

We hypothesize that the learned cost functions could enable distributed control in dense airspaces, as an alternative to completely centralized trajectory generation and air traffic control. The airplane spacing function $\bar{J}_o$ learns the relative importance of other vehicles during trajectory planning, indicating which other vehicles can be ignored during planning. The airport routing function $\bar{J}_a$ may be found to encode which areas of the airspace are unused and can be utilized by other autonomous air traffic.

Part III

Distributed Communication

Chapter 5

Learning Connectivity for Data Distribution in Robot Teams

Robot teams must often operate in harsh environments without existing communication infrastructure, requiring the formation of ad-hoc networks to exchange information. Furthermore, agent actions may take them out of direct communication range with a subset of the team so that packets must be relayed through intermediate nodes to reach their intended destination. Ad-hoc robot networks are implicitly decentralized. In spite of this, many algorithms for control of multi-robot teams operate under the assumption that low-latency, global state information necessary to coordinate agent actions can readily be disseminated among the team. While detail about how this data distribution task is accomplished is often scarce, its success is vital to the overall performance of the team. In this work, we address this challenge by providing a task-agnostic, decentralized, low-latency method for data distribution in an ad-hoc network using Graph Neural Networks. Our system enables existing centralized algorithms to be deployed in robot teams operating in harsh, real world environments.

The problem of data distribution in a team of robots bears a strong resemblance to the problem of packet routing in a mobile ad-hoc network (MANET). In our case, up-to-date global state information such as velocity, pose, or map data must be maintained at each robot so that they can choose appropriate control actions. Likewise, ad-hoc routing protocols in MANETs require each node maintain some knowledge of the state of the network so that they can choose the next hop a packet should take in its journey from source to

destination. For proactive protocols, this is often accomplished by periodically flooding the network with link state information so that each node maintains a current understanding of the network topology [84]. This process can be inefficient and strain the network with contention and packet collisions and as a result, many different flooding schemes have been developed to mitigate the so-called broadcast storm problem [85]. As is made clear in Section A, our method draws inspiration from these flooding schemes and employs them as points of comparison during evaluation.

In this work, we assume the network is used exclusively for the data distribution task in question. As such, it makes sense to skip the regular link layer overhead necessary to operate an ad-hoc routing protocol and directly consider a data distribution protocol handling the application layer traffic of the robotic system. In essence, our data distribution protocol is itself a routing protocol which circulates the state information needed by the robots in the place of network state information needed for packet routing. While conventional approaches use some form of flooding to accomplish this task, most are based on heuristics designed to minimize the age of information and network overhead [85]. There exists no clear optimal approach. Thus, we believe this problem is well suited to a data driven approach based on Graph Neural Networks paired with reinforcement learning.

Graph Neural Networks (GNNs) have shown great promise for learning from information described by a graph. Recent works have used GNNs to generate heuristic solutions to a variety of multi-robot problems, such as path planning [14, 13, 15], exploration [16], and perimeter defense [17]. In this work, a GNN is the natural choice for parameterizing the communication policy which prescribes actions based on the information exchanged over the network graph. Furthermore, GNNs offer attractive properties such as permutation invariance, making the learned solution robust to changes in graph topology between training and testing.

In order to effectively distribute information, our model must learn when and with whom to communicate. Many prior approaches use multi-agent reinforcement learning to address communication for particular cooperative control tasks, [86, 87, 88, 30]. These approaches generally do not incorporate bandwidth limitations or physical channel models, often assuming that all agents are reachable through broadcast communications, which is sometimes range-limited. Other approaches learn communication graphs to keep communication sparse [89, 90]. However, such approaches begin creating these graphs by first assuming that the relative positions of all other agents can be noisily observed [89] or are transmitted through a series of global broadcasts

[90]. In this work, we consider realistic communication conditions and thus assume no prior knowledge of other agents except for their identities and overall team size.

In this work, we propose a learned, cross-layer data distribution protocol that can be applied to any multi-agent task that relies on sharing time-sensitive local observations across a wireless network. The remainder of the chapter is organized as follows: Section A describes the wireless communication model, local data structures and the AoI minimization problem. Section B details the design of the transmission-response protocol and our usage of reinforcement learning to train the GNN architecture. Section B also describes the agent dynamics, the mission specification for the applications of our approach, and the baselines to which we compare the performance of the GNN. In Section C, we showcase our protocol's performance. The development of the GNN architecture relies on Section C of Chapter 1.

This chapter has been published as [91]. An open-source implementation of our work can be found at `https://github.com/landonbutler/Learning-Connectivity`. A video demonstration is available at `https://youtu.be/UNBvsPZIudU`.

A Problem Formulation

Wireless Communication Model. In order to reason about transmitting information in a network, it is necessary to establish a model for communication. In this work, we consider ideal wireless conditions between agents with line of sight channels and successful packet reception subject to interference from other transmitting nodes in proximity. Transmitted signals propagating through the air experience free-space path loss $\mathcal{E}_t^{i,j}$ given by:

$$\mathcal{E}_t^{i,j} = 20\log_{10}(d_t^{i,j}) + 20\log_{10}(\nu) - 147.55 \tag{5.1}$$

where $d_t^{i,j}$ is the distance between transmitting agent i and receiving agent j, and ν is the system center frequency (in Hz) [92]. This approach is consistent with wireless studies involving transmissions between aerial agents [93, 94]. From path-loss, we can define the channel gain as:

$$g_t^{i,j} = 10^{-\frac{\mathcal{E}_t^{i,j}}{10}} \tag{5.2}$$

and the signal-to-interference-plus-noise ratio (SINR) as:

$$\Gamma_t^{i,j} = \frac{\rho_t^i \cdot g_t^{i,j}}{\sigma + \sum\limits_{k \in \mathcal{A} \setminus i} \rho_t^k \cdot g_t^{k,j}} \tag{5.3}$$

where σ is additive white Gaussian noise representing ambient noise power at the receiver and ρ_t^i is the transmit power of agent i. In order for a packet transmitted from agent i to be successfully received at agent j, $\Gamma_t^{i,j}$ must exceed a SINR threshold $\hat{\Gamma}$ that takes into account the QoS of the link [95]. Defining R_t^i as the set of agents that receive a transmission from agent i at time t, we can express the probability of successful packet reception at j as:

$$p(j \in R_t^i) = \begin{cases} 1 & \Gamma_t^{i,j} \geq \hat{\Gamma} \\ 0 & \Gamma_t^{i,j} < \hat{\Gamma} \end{cases} \tag{5.4}$$

Note that in ad-hoc networks, it is common for all nodes to contend for the same medium and thus be able to decode any packet transmission for which they are in range, regardless of if they are the intended recipient. In fact, this is a foundational aspect of many flooding algorithms [84]. We follow suite by allowing agents to eavesdrop on each others transmissions.

Local Data Structures. Critical to any networking protocol is the metadata it caches for decision making. In our approach, this information forms the feature vectors consumed by our communication policy. And, since this is a data distribution task, there is an additional payload: given a set of robots $\mathcal{A}$ with clocks synchronized at time t, each robot maintains a data structure that contains agent i's knowledge about agent j indicated by $M_t^{i,j}$. This information is required by the robot to make decisions to accomplish some task. Along with the last known state, agent i also stores a timestamp for this observation of the state of agent j, $T_t^{i,j}$. Agents can observe their own local states at the current time step, $\mathbf{x}_t^i$, and update their own memory accordingly:

$$M_t^{i,i} := \mathbf{x}_t^i, \quad T_t^{i,i} := t \qquad \forall i \in \mathcal{A},\ t \geq 0 \tag{5.5}$$

Agents may make a decision to transmit their local data to others. The set S_t^i represents the intended recipients of a transmission by agent i at time t. In this work, we constrain the set of intended recipients to be at most one other agent. As outlined in the previous section, agents in proximity that transmit simultaneously run the risk of interfering with each other causing communication failures. Thus, we define f as the stochastic

communication model that determines the probability of receipt $p(j \in R_t^i)$ of agent i's transmission by agent j at time t:

$$\left\{p\left(j \in R_t^i\right)\right\}_{i \in \mathcal{A}} = f\left(\left\{S_t^i, \mathbf{x}_t^i\right\}_{i \in \mathcal{A}}\right), \tag{5.6}$$

where f is the communication model described in Eq. 5.4. We also record the timestamp at which agent i last attempted communication with agent j, $L_t^{i,j}$:

$$j \in S_t^i \quad \Longrightarrow \quad L_t^{i,j} = t. \tag{5.7}$$

As data is transmitted by agents in the team, we can track how it propagates through the network as shown in the example in Fig. 5.1. We define the parent reference notation $P_t^{i,k} = j$ to indicate that agent k first passed its information to agent j on its way to agent i. If agent i directly receives agent k's information from k, then $P_t^{i,k} = i$. When agent i receives a transmission from agent j containing agent k's state, agent i can record the link along which agent j had obtained that information, $P_t^{i,k} = P_t^{j,k}$. This record of transmission relationships describes how data flowed between agents to construct agent i's knowledge of the team's state at time t. If agent i has received agent j's transmission, then it will update its data structure if any of the received information is newer than its current data, $\forall k \in \mathcal{A}$:

$$T_t^{i,k} < T_t^{j,k}, j \in R_t^i \quad \Longrightarrow \quad (T_t^{i,k} = T_t^{j,k}) \wedge (M_t^{i,k} = M_t^{j,k}) \wedge (P_t^{i,k} = P_t^{j,k}) \tag{5.8}$$

Local Policy to Minimize Age of Information. To effectively disseminate data in the network, a communication policy is installed at every node that operates entirely off of the information outlined in the previous section. More formally, a policy π selects a recipient for its transmission using the set of information available to agent i at time t: $\left\{T_t^{i,j}, M_t^{i,j}, P_t^{i,j}, L_t^{i,j}\right\}_{j \in \mathcal{A}}$. The same local stochastic communication policy π is to be used by all agents:

$$p(k \in S_t^i) = \pi\left(\left\{T_t^{i,j}, M_t^{i,j}, P_t^{i,j}, L_t^{i,j}\right\}_{j \in \mathcal{A}}\right) \quad \forall i, k \in \mathcal{A} \tag{5.9}$$

Given the dynamic nature of robot teams, it is vital that each node maintain up-to-date information. Thus, the stochastic communication policy π needs to minimize the average age of information (AoI) across all agents, where $t - T_t^{i,j}$ is the age of information for i's knowledge of j's state. To obtain the performance criteria for

the whole system, we average over all agents, $\mathcal{A}$, the mission duration, $\mathcal{T}$, and team dynamics, $\mathcal{X}$:

$$\min_{\pi} \quad \mathbb{E}_{t\in\mathcal{T},\ i\in\mathcal{A},\ \mathbf{x}_t^i\in\mathcal{X}} \left[t - T_t^{i,j} \right] \tag{5.10}$$

The objective of the data distribution problem is to minimize the average age of information across the team. Data distribution is a cooperative task, so we can provide the team with a single reward signal, rather than assigning rewards per agent. Furthermore, the training procedure is centralized, but the policy relies on only local information, so it can be deployed in a distributed system.

We assume that the number and identities of robots in the team, $\mathcal{A}$, are known prior to mission execution. We also assume that robots share a common reference frame and a synchronized clock. Finally, we assume that communications are not bandwidth-limited relative to the size of the transmitted data, $\{T_t^{i,j}, M_t^{i,j}, P_t^{i,j}\}_{i,j\in\mathcal{A}}$. The memory required for each agent to store its knowledge of the system state is linear in the number of agents, $\mathcal{O}(|\mathcal{A}|)$, but system-wide, the memory and communication requirements are $\mathcal{O}(|\mathcal{A}|^2)$.

B Methods

Aggregation Graph Neural Networks. Our goal is to develop a policy for making communication decisions that each agent can evaluate using its local data structure. Recall that at each point in time t, every agent i has cached the set $\left\{T_t^{i,j}, M_t^{i,j}, P_t^{i,j}, L_t^{i,j}\right\}_{j\in\mathcal{A}}$. By taking a graph perspective of the data, we can structure it in a way useful for learning. To begin, we form the set of node features as: $V_t^i = \left\{T_t^{i,j}, M_t^{i,j}\right\}_{j\in\mathcal{A}}$. The parent references $P_t^{i,j}$ capture adjacency relationships in the network, describing a tree structure from the perspective of each agent i. We can define a directed edge in this tree by the ordered tuple (r_l, s_l), where r_l and s_l are the receiver and sender node, respectively, for the directed edge of index l. Then, the set of all directed edges in the graph of agent i at time t is $E_t^i = \left\{\left(P_t^{i,k}, k\right)\right\}_{k\in\mathcal{A}\setminus i}$. In this work, no input edge attributes are used. Putting it all together, we denote the graph that represents agent i's knowledge of the team at time t as $\mathcal{G}_t^i = \left\{E_t^i, V_t^i\right\}$.

Transmission-Response Protocol. The GNN outlined in the previous section is responsible for deciding who a node should attempt to communicate with, if at all, at a given timestep. In this section we detail the communication protocol that executes these plans. The protocol is divided into two main steps: a transmission phase, following the output of the GNN as one might expect, and a response phase, where certain recipients

of the transmission are able to respond. The addition of this response phase provides the transmitter with the ability to seek out information by targeting certain agents to exchange information with, a behavior that becomes valuable in the following section on reinforcement learning.

In the response phase, those agents that receive the packet for which they are the intended recipients, $\forall j$ s.t. $(j \in R_t^i) \wedge (j \in S_t^i)$, are able to respond by transmitting a message back to the set of original transmitter(s), indicated by $\bar{S}_t^j$. Note that at all times an agent can only receive at most one transmission at a time. This is a fundamental limitation in ad-hoc networks with a shared medium. Furthermore, two transmissions targeting the same destination will likely result in a packet collision with no data successfully decoded at the receiver. In the response phase, we assume that if an agent is able to respond, it will attempt to do so. Recall that agents other than those who are the designated recipients can also decode transmitted messages and update their data structure accordingly. This set of agents, called *eavesdroppers*, are described by $\forall j$ s.t. $(j \in R_t^i) \wedge (j \notin S_t^i)$.

We summarize the design of our protocol in Alg. 4. A communication time window t consists of both a transmission and a response phase. At the beginning of a window, agents first update their own state using local observations. Then, they pass their updated data structure into the local communication policy to output a set of intended recipients for their transmission. If this set consists only of the agent itself, the agent chooses not to communicate during the given transmission phase. Agents then transmit to their intended recipients. For communications that are received above the SINR threshold, agents will compare the contents of the communication to their own local data structure, updating it with any new information received. Once data structure updates from the transmission phase conclude, the response phase commences. An agent will then attempt to respond back to the agent it received a transmission from, if at all. All responses are made concurrently. Again, for successful responses received above the SINR threshold, the data structure updating procedure occurs again.

Reinforcement Learning. Finally, with the communication protocol in place, we turn to our attention to finding a solution. To solve the problem in (5.10) via reinforcement learning, we use the Proximal Policy Optimization algorithm [42] and parametrize both the policy and value functions as graph neural networks to take advantage of the modular nature of the data distribution task.

The policy and value architectures are two separate neural networks, parametrized as as non-linear GNNs described by Eq. 1.12 of Section C. Both models have f_{enc}, f_{dec} and GN as 3 layer MLPs with 64 hidden

units, and a ReLU activation only after the first two layers. For the policy, f_{out} is a linear function that reduces the high-dimensional latent space vectors to the required low-dimensional output, the logits of a Boltzmann distribution. Using the Gumbel-softmax, we use the logits to generate a discrete distribution over potential actions. At each timestep, each robot samples from this distribution to decide the recipient of its transmission. Also included as an action is the option for the agent not to transmit at the current timestep. For the value function, f_{out} outputs 1 scalar per agent, which are summed to compute the team's value estimate.

Unless noted otherwise, we use a receptive field of 5 across all experiments. Each model was trained using an open source implementation of PPO [96] using 2×10^6 observations. We use an Adam optimizer with step size 1×10^{-4} that is decayed by a factor of 0.95 for every 500 steps, and a batch size of 64. All input features were normalized by the mission area and the duration of the mission where relevant prior to being input to the neural network.

Simulations. To evaluate our data distribution system, we conducted simulations with a robot team performing three main tasks. The first two tasks serve to establish performance baselines and involve the agents passing information while stationary and while moving with random velocities. In the third task we use our system in a flocking maneuver, where each agent sets its own velocity according to its knowledge of the velocity of the rest of the team. In this case, global information about the velocity of each member of the team must be distributed in a timely manner so that the team converges on a common velocity before any agents can wander off. In each case, we report the age of information achieved by our system compared with a suite of other data distribution approaches outlined in Section B.

Agent Dynamics. In simulation we treat each robot as a point mass in $\mathbb{R}^2$ with first-order dynamics described by:

$$\mathbf{p}_{t+1}^i = \mathbf{p}_t^i + \mathbf{v}_t^i \Delta t, \quad \mathbf{v}_{t+1}^i = \mathbf{v}_t^i + \mathbf{u}_t^i \Delta t \tag{5.11}$$

where for agent i at time t, $\mathbf{p}_t^i \in \mathbb{R}^2$ is the agent's position and $\mathbf{v}_t^i \in \mathbb{R}^2$ is the agent's velocity. An agent's local state is described by the 4-dimensional column vector $\mathbf{x}_t^i := [\mathbf{p}_t^i; \mathbf{v}_t^i]$ for agent i at time t. This is the local state the agent transmit to others.

For the second task involving random motion, we use control inputs sampled uniformly according to $\mathbf{u}_t^i \sim \mathcal{N}(0, A_{max}/3)$. The control inputs for this task do not rely on the performance of the data distribution

algorithm. In addition, agent velocities are clipped to the interval $(-V_{max}, V_{max})$ and velocities are also reflected when agents come into contact with the boundaries of the mission area, $(-P_{max}, P_{max})$, described in the following section.

In the flocking task, agents seek consensus in velocities, as can be seen in example in Fig. 5.2. This task is sensitive to latency in the communication among fast-moving agents, so data distribution is essential for it success [30]. Agent i runs a controller based on its observations of the team after the initial timestep $T_0^{i,j}$, $\mathcal{F}_t^i = \{j \in \mathcal{A} \text{ s.t. } T^{i,j} > T_0^{i,j}\}$, which computes the difference between the agent's current velocity and the average velocities among all observed agents:

$$\mathbf{u}_t^i := \frac{1}{|\mathcal{F}_t^i|} \sum_{j \in \mathcal{F}_t^i} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \left(M_t^{i,j} - \mathbf{x}_t^i \right) \tag{5.12}$$

This is a variant of the task proposed in [52] but without control of agent spacing.

For a model trained to execute data distribution for flocking, we can quantify not only the age of information cost in (5.10), but we can also benchmark performance on the flocking task itself [30], quantifying the variance in agents' velocities:

$$C = \frac{1}{|\mathcal{A}|} \sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{A}} \left\| \mathbf{v}_t^i - \frac{1}{|\mathcal{A}|} \left[\sum_{j \in \mathcal{A}} \mathbf{v}_t^j \right] \right\|^2 \tag{5.13}$$

These two performance metrics for the flocking task are heavily correlated, but this may not be true for other tasks, in which low-latency information may be less important.

Mission Specification. The team of 40 robots operates in a mission area of 1 km^2, where robots' initial positions were generated uniformly at random within the mission space. For teams of other sizes, the mission area is changed to maintain a fixed agent density of 40 agents per 1 km^2. For mobile agents, the velocity is provided as a ratio of the maximum distance the agent could travel across the mission area during the mission. The default velocity ratio is 0.15, corresponding to $V_{max} = 3$ m/s. The maximum acceleration is set to be $A_{max} = 20$m/s^2 for all experiments with mobile teams.

We allot a communication time window of 100ms, in which both a request and response is made. In the case of random flooding, two requests are made. Each mission is 500 steps long, allowing 500 rounds of communication among agents. The default maximum communication range for agents is 250 meters, which we plot in normalized units as 0.25 of the mission distance. Team positions were re-initialized until

the communication graph had algebraic connectivity at the initial timestep, assuming a disk communication model with a range of 250 m. Across all experiments, we maintain a SINR threshold of 1 dBm, an additive white Gaussian noise of -50 dBm, and a system center frequency of 2.4 GHz.

Baselines. Data distribution is a regular part of many ad-hoc routing protocols. Thus, we compare our learned approach to three representative baselines inspired by such protocols: random flooding, round robin, and minimum-spanning tree.

Flooding is perhaps the most popular data distribution method used in ad-hoc routing protocols. The simplest form involves repeatedly rebroadcasting a message originating at a source node until it has reached every agent in the network. More advanced variants seek to improve inefficiencies by introducing rules governing which nodes should participate in rebroadcasting. In our simulations, we employ random flooding, which instructs agents to rebroadcast a message with some probability p [85]. While flooding in ad-hoc protocols often happens periodically to update network topology information, data distribution in the context of robot teams happens continuously, as the state information of each agent is constantly evolving. Thus, for our problem, the choice a node makes is whether to broadcast the latest information it has gathered, including what it has received from neighbors (i.e. rebroadcasting), or stay silent and allow other nodes to transmit. Since each timestep of our protocol involves a transmission and response, we allow random flooding to complete two iterations over the same period.

The second method we compare against is round robin. In this approach, a central agent is chosen as a base station and at each timestep, one other agent exchanges data structures with it, reminiscent of a time-division multiple access approach [97]. Unlike our approach and random flooding, round robin requires centralized coordination to identify the base station agent and schedule communications of other agents with the base station. Round robin follows the same transmit response used by our approach with S_t^i always populated by the base station.

Finally, we also implement a Minimum Spanning Tree (MST) baseline where each agent attempts to exchange information with their parent in the spanning tree with probability p, remaining silent otherwise. This baseline seeks to capitalize off the fact that a MST minimizes the total edge length required to connect all the agents in the network. As such, the edges of the tree offer an efficient way to pass information throughout the network. One limitation of this approach is that it requires global knowledge of all agents' locations from the start of the episode. At a given timestep, the transmissions between child and parent follow the

transmit-response format used by our method.

C Results

First, we highlight the impact of the GNN's receptive field on its performance on the data distribution task, and characterize the impact of transmission power on the overall task difficulty. Next, we examine how the GNN can generalize to larger team sizes. Finally, we benchmark the GNN as the data distribution for a latency-sensitive flocking task.

Stationary Teams. Our first experiment investigates the effect of receptive field on the performance of the GNN controller as shown in Fig. 5.3. We observe a general trend of improved performance with an increase in the receptive field of the architecture, which allows the model to reason about clusters of agents in its buffer. The GNNs with receptive fields of two or greater surpass the performance of Round Robin, MST and Random Flooding baselines.

Fig. 5.4 demonstrates the effect of varying the transmission power ratio. As the maximum transmission range increases, the problem becomes exponentially easier. However, many multi-robot teams operate in settings where hardware or environmental constraints restrict transmission distance. In these settings, especially when agents can only communicate at 25% of the mission distance, our protocol performs best, having the greatest margin from the baseline approaches. The introduction of structure in how data propagated throughout the network allows our protocol to establish known routes of data flow. Instead of relying on direct communication with agents from the other side of the network, our protocol learns to leverage the established routes and often only needs short-range communications to access these.

Mobile Teams. Next, we examine the performance of the GNN in data distribution for mobile teams. Fig. 5.5 demonstrates the generalization of a model trained on 40 mobile agents (GNN Generalization) to team sizes of 10 up to 80. For comparison, we also show the performance of a model trained on the corresponding team size (GNN Trained). We find that both models exceed the baselines for large team sizes. The transference property of Graph Neural Networks allows the model trained on a team of 40 to scale to teams of up to 80. With team sizes greater than or equal to 80 agents, we find that the $\mathcal{O}(|\mathcal{A}|^2)$ memory requirements make training difficult, and our generalized model begins to perform best.

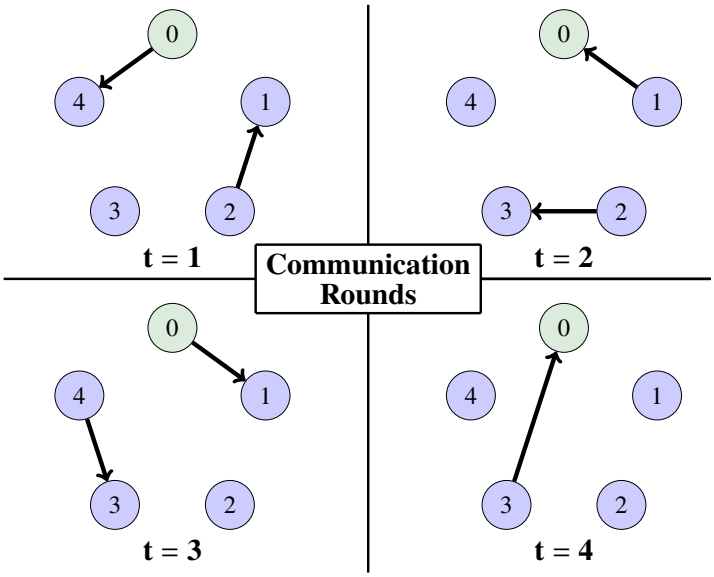
Fig. 5.6 examines the flocking application and compares GNN models trained with the task-specific

velocity variance cost (3.5) against models trained with the age of information cost (5.10). Models trained with Age of Information cost perform better on the flocking task than models trained via the variance of velocities cost because the Age of Information provides a more direct signal for communication decisions. For using reinforcement learning for other problems with a data distribution component, age of information may be a highly-informative reward signal that could be used in conjunction with the task-specific reward function. We observe that at higher velocities, the performance of both GNN controllers begins to stray away from the baselines because the initial route discovery phase is more challenging and agents may stray completely out of range before establishing communications. Providing information about the initial configuration of the team may be able to allow the GNN to improve performance from the initial time step.

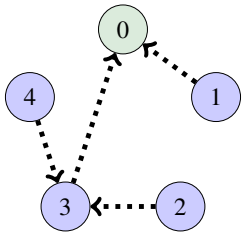
D Summary

This work introduces a task-agnostic, decentralized method for data distribution in ad-hoc networks. We demonstrate that maintaining routes of data dissemination can help agents reason about multi-hop communication, resulting in more up-to-date network information as compared to industry-standard approaches. This performance extends to mobile robot teams, and transfers well for larger team sizes and the flocking task. We envision our protocol being applied to any collaborative multi-agent task reliant on sharing time-sensitive local observations across a wireless network.

In future work, we would like to run physical experiments with asynchronous execution. We believe another natural extension of this project is to not only consider when and with whom to communicate, but also what to communicate. At each time step, each agent would decide which portions of their data structures are most essential to communicate for mission success. Lastly, the introduction of Graph Recurrent Neural Networks (GRNNs) to our system would assist in handling time-varying communication needs.



Agent 0's Local Data Structure				
ID	**TS**	**State**	**Parent**	**LC**
0	4	$M_4^{0,0}$		
1	2	$M_4^{0,1}$	0	3
2	2	$M_4^{0,2}$	3	
3	4	$M_4^{0,3}$	0	
4	3	$M_4^{0,4}$	3	1



Tree Representation of Agent 0's Local Data Structure at t = 4

Figure 5.1. After four rounds of communication, agent 0 receives data from all other agents. Because parent references are also transmitted, agent 0 is able to use its data structure to reconstruct the spanning tree consisting of the most recent information of other agents in the network. The table heading abbreviations are: TS - Last Observed Timestep, ($T_t^{i,j}$), State - Agent State Information ($M_t^{i,j}$), Parent - Data Flow Parent, LC - Last Attempted Communication.

Algorithm 4 Data Distribution Protocol

Require: $\mathcal{A}, \pi, f, \Delta t$

while true do

Transmission phase begins, $t \Leftarrow t + \Delta t$

Agents update current state in local data structures

$$M_t^{i,i} = \mathbf{x}_t^i, T_t^{i,i} = t \quad \forall i \in \mathcal{A},\ t \geq 0$$

Agents evaluate transmission policy using local data structures

$$p(k \in S_t^i) = \pi\left(\{T_t^{i,j}, M_t^{i,j}, P_t^{i,j}, L_t^{i,j}\}_{j \in \mathcal{A}}\right) \quad \forall i, k \in \mathcal{A}$$

Transmissions are subject to interference

$$\{p(j \in R_t^i)\}_{i \in \mathcal{A}} = f\left(\{S_t^i, \mathbf{x}_t^i\}_{i \in \mathcal{A}}\right)$$

Agents update local data structures, $\forall i, k \in \mathcal{A}, j \in R_t^i$

$$T_t^{i,k} < T_t^{j,k} \quad \Longrightarrow \quad (T_t^{i,k} = T_t^{j,k}) \wedge (M_t^{i,k} = M_t^{j,k}) \wedge (P_t^{i,k} = P_t^{j,k})$$

Response phase begins, $t \Leftarrow t + \Delta t$

Recipients transmit responses

$$(j \in R_t^i) \wedge (j \in S_t^i) \quad \Longrightarrow \quad i \in \bar{S}_t^j, \quad \forall j \in \mathcal{A}$$

Responses are subject to interference

$$\{p(i \in \bar{R}_t^j)\}_{j \in \mathcal{A}} = f\left(\{\bar{S}_t^j, \mathbf{x}_t^j\}_{j \in \mathcal{A}}\right)$$

Agents update local data structures, $\forall i, k \in \mathcal{A}, j \in \bar{R}_t^i$

$$T_t^{i,k} < T_t^{j,k} \quad \Longrightarrow \quad (T_t^{i,k} = T_t^{j,k}) \wedge (M_t^{i,k} = M_t^{j,k}) \wedge (P_t^{i,k} = P_t^{j,k})$$

end while

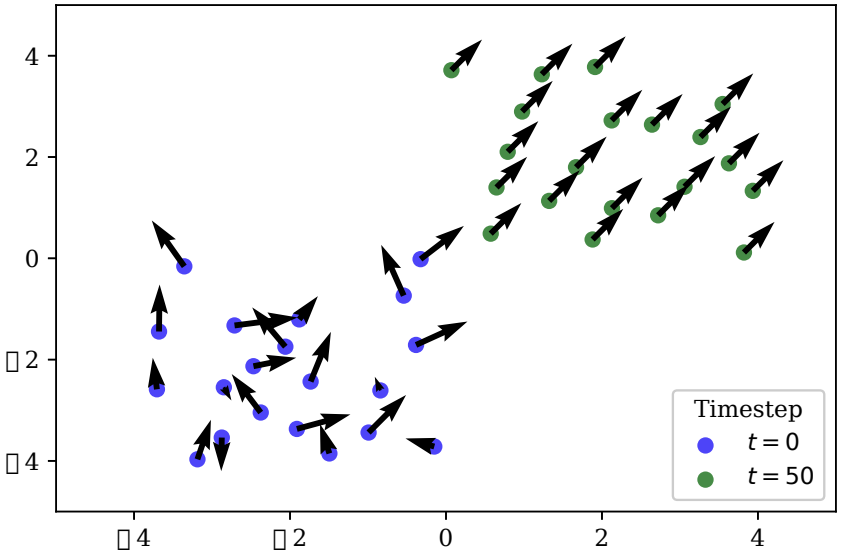


Figure 5.2. After starting with random velocities, agents communicate and eventually come to a global consensus.

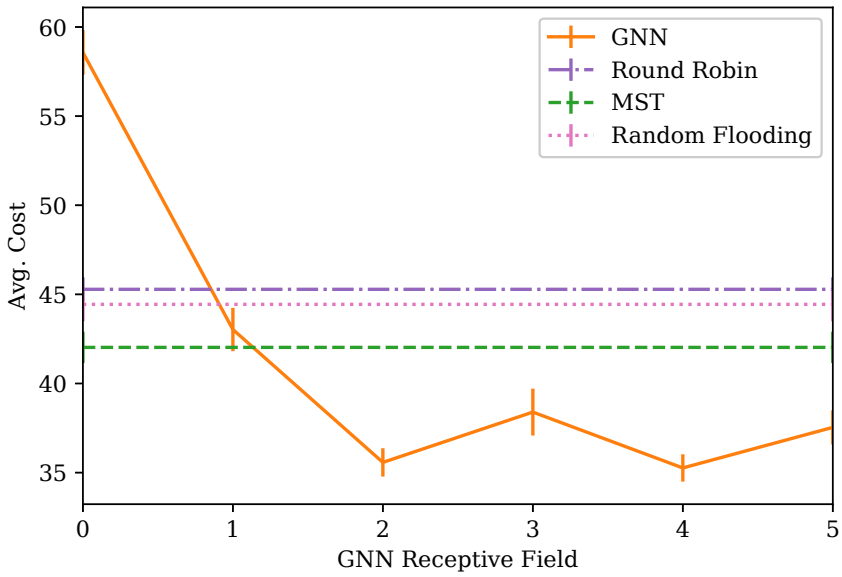


Figure 5.3. For stationary agents, as the GNN receptive field is increased for the linear and non-linear models, AoI performance improves and exceeds MST and Round Robin.

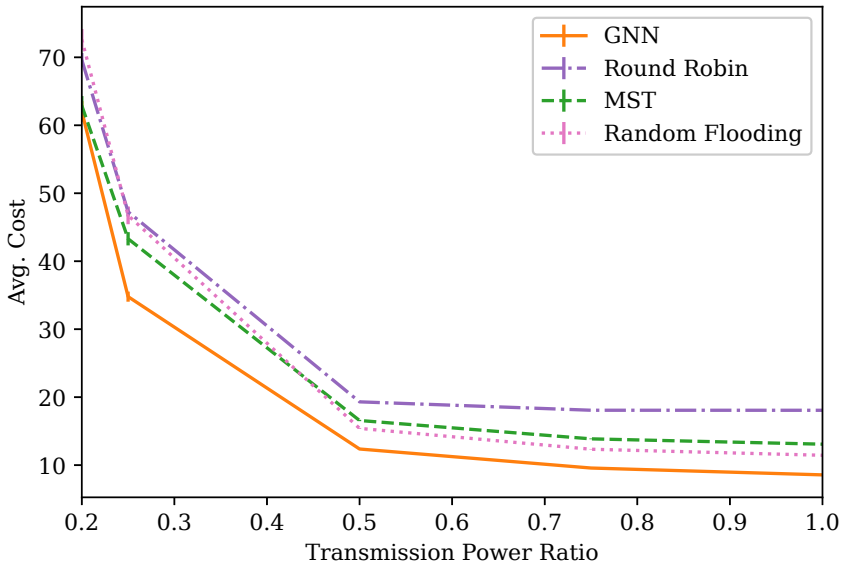


Figure 5.4. For stationary agents, as agent transmission power increases, the data distribution task incurs lower latency on average for both the baseline controllers and the learner.

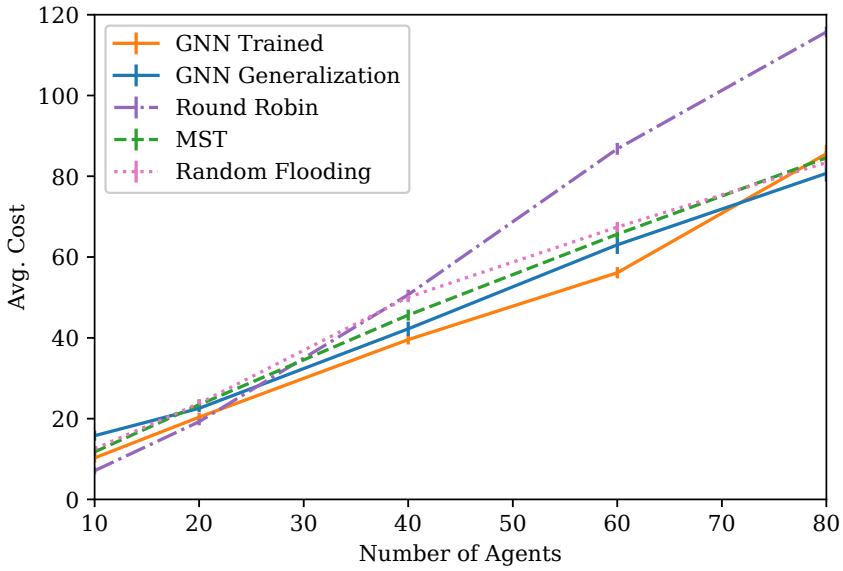
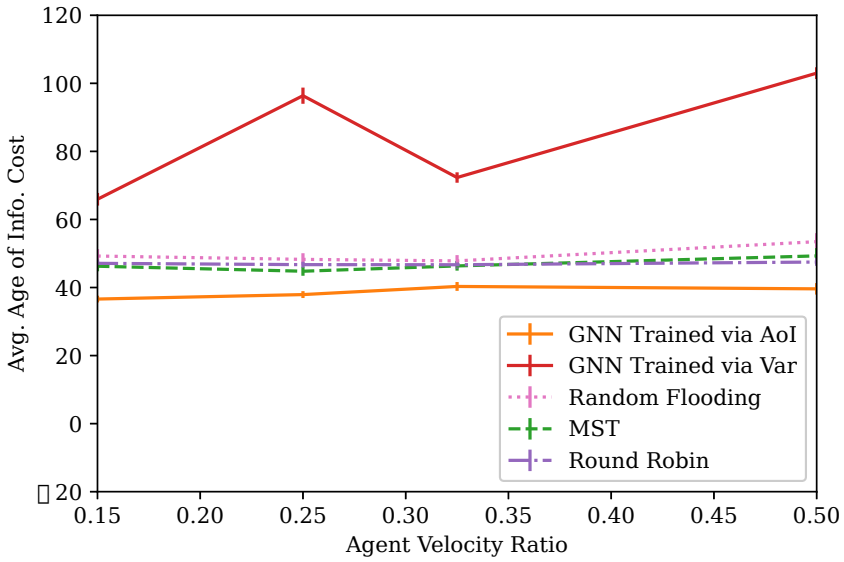
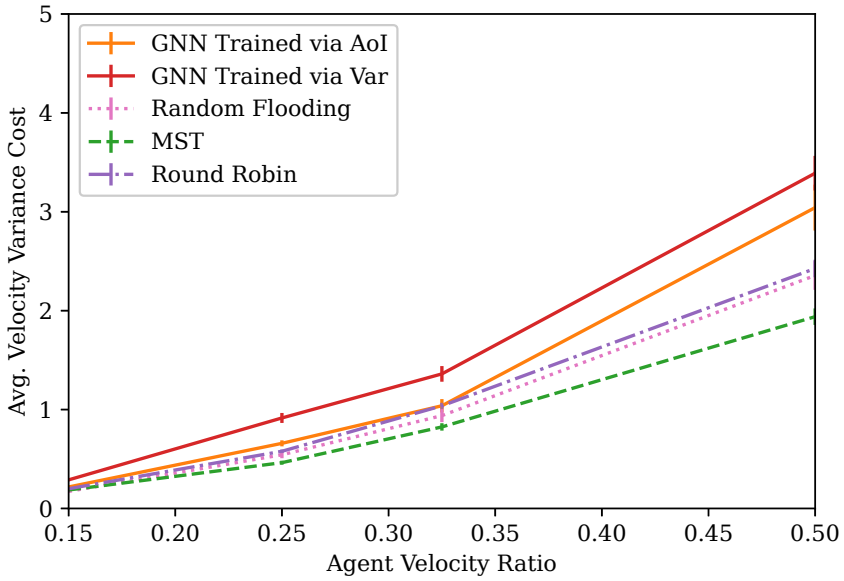


Figure 5.5. GNN models trained on teams of 40 agents performing a random walk with a velocity ratio of 0.15 generalize well to larger team sizes.



(a) Testing Age of Information Cost



(b) Testing Velocity Variance Cost

Figure 5.6. As agent velocity increases in the flocking task, the learner's performance begins to decline. Testing the velocity variance, the controller trained using the AoI reward performs better than the controller trained using Velocity Variance cost.

Part IV

Towards Distributed Learning

Chapter 6

Kernel Q-Learning in Continuous Markov Decision Problems

Markov Decision Problems offer a flexible framework to address sequential decision making tasks under uncertainty [98], and have gained broad interest in robotics [99], control [100], and artificial intelligence [101]. Despite this surge of interest, few works in reinforcement learning address the computational difficulties associated with continuous state and action spaces in a principled way that guarantees convergence. The goal of this work is to develop new reinforcement learning tools for continuous problems which are provably stable and whose complexity is at-worst moderate.

In the development of stochastic methods for reinforcement learning, one may attempt to estimate the transition density of the Markov Decision Process (MDP) (model-based [102]), perform gradient descent on the value function with respect to the policy (direct policy search [103]), and pursue value function based (model-free [104, 105]) methods which exploit structural properties of the setting to derive fixed point problems called *Bellman equations*. We adopt the latter approach in this work [106], motivated by the fact that an action-value function tells us both how to find a policy and how to evaluate it in terms of the performance metric we have defined, and that a value function encapsulates structural properties of the relationship between states, actions, and rewards.

It is well-known that approaches featuring the "deadly triad" [101] of function approximation, bootstrapping (e.g. temporal-difference learning), and off-policy training are in danger of divergence, and the most

well-understood techniques for ensuring convergence in a stochastic gradient descent context are those based on Gradient Temporal Difference (GTD) [107]. Though the final algorithm looks similar, our approach could be considered as an alternative formulation and analysis of the GTD algorithms centered on continuous state and action spaces with nonlinear function approximation.

To frame our approach, consider the fixed point problem defined by Bellman's optimality equation [108]. When the state and action spaces are finite and small enough that expectations are computable, fixed point iterations may be used. When this fails to hold, stochastic fixed point methods, namely, Q-learning [105], may be used, whose convergence may be addressed with asynchronous stochastic approximation theory [109, 110]. This approach is only valid when the action-value (or Q) function may be represented as a matrix. However, for continuous the state and action spaces, the Q-function instead belongs to a generic function space.

In particular, to solve the fixed point problem defined by Bellman's optimality equation when spaces are continuous, one must surmount the fact that it is defined for infinitely many unknowns, one example of Bellman's curse of dimensionality [108]. Efforts to sidestep this issue assume that the Q-function admits a finite parameterization, such as a linear [107] or nonlinear [111] basis expansion, is defined by a neural network [2], or that it belongs to a reproducing kernel Hilbert Space (RKHS) [112]. In this work, we adopt the latter nonparametric approach, motivated by the fact that combining fixed point iterations with different parameterizations may cause divergence [113, 114]. To ensure algorithm stability, we tie Q-function parameterization to the stochastic update [115].

Our main result is a memory-efficient, non-parametric, stochastic method that converges to a fixed point of the Bellman optimality operator almost surely when it belongs to a RKHS. We obtain this result by reformulating the Bellman optimality equation as a nested stochastic program (Section A) This problem has been addressed in finite settings with stochastic *quasi-gradient* (SQG) methods [116] which use two time-scale stochastic approximation to mitigate the fact that the objective's stochastic gradient not available due to its dependence on a *second expectation*, which is referred to as the double sampling problem in [107].

Here, we use a non-parametric generalization of SQG for Q-learning in infinite MDPs (Section B), motivated by its success for policy evaluation in finite [107] and infinite MDPs [117]. However, a function in a RKHS has comparable complexity to the number of training samples processed, which is in general infinite, an issue is often ignored in kernel methods for Markov decision problems [118, 119]. We address this bottleneck (the curse of kernelization) by requiring memory efficiency in both the function sample path and in

its limit through the use of sparse projections which are constructed greedily via matching pursuit [120, 121], akin to [122, 117]. Greedy compression is appropriate since (a) kernel matrices induced by arbitrary data streams will likely become ill-conditioned and hence violate assumptions required by convex methods [123], and (b) parsimony is more important than exact recovery as the SQG iterates are not the target signal but rather a stepping stone to Bellman fixed point. Rather than unsupervised forgetting [124], we tie the projection-induced error to guarantee stochastic descent [122], only keeping dictionary points needed for convergence. We note this projection operator was originally developed based upon projecting stochastic gradients, and whose performance may be related to a stochastic descent criterion [122]. Here we consider a generalization based upon stochastic quasi-gradients and tune projection-induced error to ensure *stochastic quasi-descent criterion*, as necessitated by the two time-scale nature of compositional approaches to Q-learning.

As a result, we conduct functional SQG descent via sparse projections of the SQG. This maintains a moderate-complexity sample path exactly towards Q^*, which may be made arbitrarily close to a Bellman fixed point by decreasing the regularizer. In contrast to the convex structure in [117], the Bellman optimality equation induces a non-convex cost functional, which requires us to generalize the relationship between SQG for non-convex objectives and coupled supermartingales in [125] to RKHSs. In doing so, we establish that the sparse projected SQG sequence converges almost surely (Theorem 1) to the Bellman fixed point with decreasing learning rates (Section D) and to a small Bellman error whose magnitude depends on the learning rates when learning rates are held constant (Theorem 2). Use of constant learning rates allows us to further guarantee that the memory of the learned Q function remains under control. Moreover, on Continuous Mountain Car [126] and the Inverted Pendulum [127], we observe that our learned action-value function attains a favorable trade-off between memory efficiency and Bellman error, which then yields a policy whose performance is competitive with the state of the art in terms of episode average reward accumulation. Results for Pendulum are given in Appendix C

In contrast to [117], in addition to non-convexity, challenges related to the explore-exploit tradeoff and the need to optimize the Q function in the inner-loop of the algorithm are present. The first we experimentally overcome using replay buffers and approximate greedy policies, whereas the ability to tractably deal with the latter is unique to the RKHS parameterization. In principle, one could employ [125] to solve Q learning over continuous spaces, using a linear basis expansion over a sufficiently large number of features. However, for problems where the number of episodes required for training is large, one must use a fine-resolution grid over

the entire state and action space. In theory, the resolution must go to null as the number of samples becomes large, so that the number of features approaches infinity. Doing so, unfortunately, results in retaining many extraneous features in the function parameterization and a loss of interpretability. We these tradeoffs in Sec. E. In particular, in Table 6.1 one may observe that "Budgeted" [128] and "Discretized" approaches [125] yield a less favorable tradeoff of complexity, convergence accuracy, and reward accumulation relative to variants of KQ Learning.

This chapter was first published as [129], with additional theoretical and experimental results provided in [130]. An open-source implementation of our work can be found at `https://github.com/katetolstaya/kernelrl`.

A Markov Decision Processes

We model an autonomous agent in a continuous space as a Markov Decision Process (MDP) with continuous states $\mathbf{s} \in \mathcal{S} \subset \mathbb{R}^p$ and actions $\mathbf{a} \in \mathcal{A} \subset \mathbb{R}^q$. When in state $\mathbf{s}$ and taking action $\mathbf{a}$, a random transition to state $\mathbf{s}'$ occurs according to the conditional probability density $\mathbb{P}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$. After the agent transitions to a particular $\mathbf{s}'$ from $\mathbf{s}$, the MDP assigns an instantaneous reward $r(\mathbf{s}, \mathbf{a}, \mathbf{s}')$, where the reward function is a map $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ whose range is assumed bounded throughout by a constant $R < \infty$.

In Markov Decision problems, the goal is to find the sequence of actions defined by $\mathcal{F}_t$-adapted processes $\{\mathbf{a}_t\}_{t=0}^{\infty}$ so as to maximize the infinite horizon accumulation of rewards, i.e., the value function: $V(\mathbf{s}, \{\mathbf{a}_t\}_{t=0}^{\infty}) := \mathbb{E}_{\mathbf{s}'}[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t) \mid \mathbf{s}_0 = \mathbf{s}, \{\mathbf{a}_t\}_{t=0}^{\infty}]$. By $\mathcal{F}_t$-adapted process, we mean each $\mathbf{a}_t$ is selected only on the basis of past rewards $r(\mathbf{s}_u, \mathbf{a}_u, \mathbf{s}'_u)$, states $\mathbf{s}_u$, and actions $\mathbf{a}_u$ for $u < t$. The expectation is an integral over the Markov transition dynamics conditioned on a fixed action sequence and initial state. Moreover, $\gamma^t \in (0, 1)$ is a discount factor that determines the degree of farsightedness of the policy. The action-value function $Q(\mathbf{s}, \mathbf{a})$ is the conditional mean of the value function given the initial action $\mathbf{a}_0 = \mathbf{a}$:

$$Q(\mathbf{s}, \mathbf{a}, \{\mathbf{a}_t\}_{t=1}^{\infty}) := \mathbb{E}_{\mathbf{s}'}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t) \mid \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}, \{\mathbf{a}_t\}_{t=1}^{\infty}\right] \tag{6.1}$$

We define $Q^*(\mathbf{s}, \mathbf{a})$ as the maximum of (6.1) with respect to the action sequence. The reason for defining

action-value functions is that the optimal Q^* yields the optimal policy π^* as

$$\pi^*(\mathbf{s}) = \operatorname*{argmax}_{\mathbf{a}} Q^*(\mathbf{s}, \mathbf{a}) \ . \tag{6.2}$$

where a policy is a map from states to actions: π: $\mathcal{S} \to \mathcal{A}$. Thus, finding Q^* solves the MDP. Value-function based approaches to MDPs reformulate (6.2) by shifting the index of the summand in (6.1) by one, use the time invariance of the Markov transition kernel, and the homogeneity of the summand, to derive the Bellman optimality equation:

$$Q^*(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}'}\Big[r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma \max_{\mathbf{a}'} Q^*(\mathbf{s}', \mathbf{a}') \,\Big|\, \mathbf{s}, \mathbf{a} \Big]. \tag{6.3}$$

The expectation is taken with respect to the conditional distribution $\mathbb{P}(d\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$ of the state $\mathbf{s}'$ given the state action pair $(\mathbf{s}, \mathbf{a})$. The right side of (6.3) defines the Bellman optimality operator $\mathscr{B}^*$: $\mathcal{B}(\mathcal{S}\times\mathcal{A}) \to \mathcal{B}(\mathcal{S}\times\mathcal{A})$ over $\mathcal{B}(\mathcal{S} \times \mathcal{A})$, the space of bounded continuous action-value functions Q: $\mathcal{B}(\mathcal{S} \times \mathcal{A}) \to \mathbb{R}$:

$$(\mathscr{B}^* Q)(\mathbf{s}, \mathbf{a}) := \int_{\mathcal{S}} \Big[r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}') \Big] \mathbb{P}(d\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \ . \tag{6.4}$$

[100] [Proposition 5.2] establishes that the fixed point of (6.4) is the optimal action-value function Q^*. Hence we seek to compute the fixed point of (6.4) for all $(\mathbf{s}, \mathbf{a}) \in \mathcal{S} \times \mathcal{A}$.

Compositional Stochastic Optimization. The functional fixed point equation in (6.3) has to be simultaneously satisfied for all state action pairs $(\mathbf{s}, \mathbf{a})$. Alternatively, we can integrate (6.3) over the joint distribution of the random pair $(\mathbf{s}, \mathbf{a})$ which we assume forms an irreducible Markov chain with unique stationary probability distribution to write a nested stochastic optimization problem [125, 122, 117]. To do so, begin by defining the function

$$f(Q; \mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}'}\Big[r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}') - Q(\mathbf{s}, \mathbf{a}) \,\Big|\, \mathbf{s}, \mathbf{a} \Big], \tag{6.5}$$

and consider the ergodic stationary distribution of the Markov chain $\mathbb{P}(d\mathbf{s}, d\mathbf{a})$ w.r.t. $(\mathbf{s}, \mathbf{a})$ to define [*]

$$L(Q) = \frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a}}\Big[f^2(Q; \mathbf{s}, \mathbf{a}) \Big]. \tag{6.6}$$

[*]That $\mathbb{P}(d\mathbf{s}, d\mathbf{a})$ defines the ergodic stationary distribution of the Markov chain over the process $(\mathbf{s}, \mathbf{a})$ ensures that considering Galerkin relaxation (6.7) of (6.3) is sufficient. See [131][Sec. 2.2] for further details.

Comparing (6.5) with (6.3) permits concluding that $\mathbf{Q}^*$ is the unique function that makes $f(Q; \mathbf{s}, \mathbf{a}) = 0$ for all $(\mathbf{s}, \mathbf{a})$. It then follows that $\mathbf{Q}^*$ is the only function that makes the functional in (6.6) take the value $L(Q) = 0$. As this functional is also nonnegative, we can write

$$Q^* = \operatorname*{argmin}_{Q \in \mathcal{B}(\mathcal{S} \times \mathcal{A})} L(Q) \ . \tag{6.7}$$

Computation of the optimal policy is thus equivalent to solving the optimization problem in (6.7). This requires a difficult search over all bounded continuous functions $\mathcal{B}(\mathcal{S} \times \mathcal{A})$. We reduce this difficulty through a hypothesis on the function class.

Reproducing Kernel Hilbert Spaces. We restrict $\mathcal{B}(\mathcal{S} \times \mathcal{A})$ to a Hilbert space $\mathcal{H}$ equipped with a reproducing kernel, an inner product-like map $\kappa : (\mathcal{S} \times \mathcal{A}) \times (\mathcal{S} \times \mathcal{A}) \to \mathbb{R}$ such that

$$\text{(i) } \langle f, \kappa((\mathbf{s}, \mathbf{a}), \cdot) \rangle_{\mathcal{H}} = f(\mathbf{s}, \mathbf{a}), \qquad \text{(ii) } \mathcal{H} = \overline{\operatorname{span}\{\kappa((\mathbf{s}, \mathbf{a}), \cdot)\}} \tag{6.8}$$

In (6.8), property (i) is called the reproducing property, and is derived from Riesz Representation Theorems [132]. Replacing f by $\kappa((\mathbf{s}', \mathbf{a}'), \cdot)$ in (6.8) (i) yields the expression $\langle \kappa((\mathbf{s}', \mathbf{a}'), \cdot), \kappa((\mathbf{s}, \mathbf{a}), \cdot) \rangle_{\mathcal{H}} = \kappa((\mathbf{s}', \mathbf{a}'), (\mathbf{s}, \mathbf{a}))$, the origin of the term "reproducing kernel." Moreover, property (6.8) (ii) states that functions $f \in \mathcal{H}$ admit a basis expansion in terms of kernel evaluations (6.9). These spaces are called reproducing kernel Hilbert spaces (RKHSs).

We may apply the Representer Theorem to transform the functional problem into a parametric one [133, 134]. In the RKHS, the optimal Q function takes the following form

$$Q(\mathbf{s}, \mathbf{a}) = \sum_{n=1}^{N} w_n \kappa((\mathbf{s}_n, \mathbf{a}_n), (\mathbf{s}, \mathbf{a})) \tag{6.9}$$

where $(\mathbf{s}_n, \mathbf{a}_n)$ is a sample of state-action pairs $(\mathbf{s}, \mathbf{a})$ and w_n is a scalar weight. $Q \in \mathcal{H}$ is an expansion of kernel evaluations only at observed samples.

One complication of the restriction $\mathcal{B}(\mathcal{S} \times \mathcal{A})$ to the RKHS $\mathcal{H}$ is that this setting requires the cost to be differentiable with Lipschitz gradients, but the definition of $L(Q)$ [cf. (6.6)] defined by Bellman's equation (6.4) is non-differentiable due to the presence of the maximization over the Q function. This issue may be addressed by either operating with approximate smoothed gradients of a non-differentiable function [135]

or by approximating the non-smooth cost by a smooth one. We adopt the latter approach by replacing the $\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}')$ term in (6.6) by the softmax over continuous range $\mathcal{A}$, i.e.

$$\text{softmax}_{\mathbf{a}' \in \mathcal{A}} Q(\mathbf{s}', \mathbf{a}') = \frac{1}{\eta} \log \int_{\mathbf{a}' \in \mathcal{A}} e^{\eta Q(\mathbf{s}', \mathbf{a}')} d\mathbf{a}' \tag{6.10}$$

and define the η-smoothed ($\eta > 0$ is a parameter) cost $L(Q)$ as the one where the softmax (6.10) in lieu of the hard maximum in (6.6). Subsequently, we focus on smoothed cost $L(Q)$.

In this work, we restrict the kernel used to be in the family of universal kernels, such as a Gaussian/Radial Basis Function (RBF) kernel with constant diagonal covariance Σ,

$$\kappa((\mathbf{s}, \mathbf{a}), (\mathbf{s}', \mathbf{a}')) = \exp\left\{-\frac{1}{2}\left((\mathbf{s}, \mathbf{a}) - (\mathbf{s}', \mathbf{a}')\right) \Sigma \left((\mathbf{s}, \mathbf{a}) - (\mathbf{s}', \mathbf{a}')\right)^T\right\} \tag{6.11}$$

motivated by the fact that a continuous function over a compact set may be approximated uniformly by a function in a RKHS equipped with a universal kernel [136].

To apply the Representer Theorem, we require the cost to be coercive in Q [134], which may be satisfied through use of a Hilbert-norm regularizer, so we define the regularized cost functional $J(Q) = L(Q) + (\lambda/2)\|Q\|^2_{\mathcal{H}}$ and solve the regularized problem (6.7), i.e.

$$Q^* = \underset{Q \in \mathcal{H}}{\text{argmin}}\, J(Q) = \underset{Q \in \mathcal{H}}{\text{argmin}}\, L(Q) + \frac{\lambda}{2}\|Q\|^2_{\mathcal{H}}. \tag{6.12}$$

Thus, finding a locally optimal action-value function in an MDP amounts to solving the RKHS-valued compositional stochastic program with a non-convex objective defined by the Bellman optimality equation (6.4). This action-value function can then be used to obtain the optimal policy (6.2). Next we turn to iterative stochastic methods to solve (6.12).

We point out that this is a step back from the original intent of solving (6.7) to then find optimal policies π^* using (6.2). This is the case because the assumption we have made about Q^* being representable in the RKHS $\mathcal{H}$ need not be true. More importantly, the functional $J(Q)$ is not convex in Q and there is no guarantee that a local minimum of $J(Q)$ will be the optimal policy Q^*. This is a significant difference relative to policy evaluation problems [117]. In particular, the gap between the optimizer of $L(Q)$ and $J(Q)$ comes from both nonconvexity, where it can be interpreted as a form of estimation error (bias), and the hypothesis

on the function class, a form of approximation error (variance) [137]. However, when a universal kernel is used [136], the approximation error vanishes. On the other hand, overcoming nonconvexity in Q learning over continuous spaces is challenging, and requires investigation of random perturbations of search directions [138, 139], which is left to future work.

B Stochastic Quasi-Gradient Method

To solve (6.12), we propose applying a functional variant of stochastic quasi-gradient (SQG) descent to the loss function $J(Q)$. The reasoning for this approach rather than a stochastic gradient method is the nested expectations cause the functional stochastic gradient to be still dependent on a second expectation which is not computable, and SQG circumvents this issue. Then, we apply the Representer Theorem (6.9) ("kernel trick") to obtain a parameterization of this optimization sequence, which has per-iteration complexity. We then mitigate this untenable complexity growth while preserving optimality using greedy compressive methods, inspired by [122, 117].

To find a stationary point of (6.12) we use quasi-gradients $\nabla_Q J(Q)$ of the functional $J(Q)$ relative to the function Q in an iterative process. We introduce an iteration index t and let Q_t be the estimate of the stationary point at iteration t. Consider a random state action pair $(\mathbf{s}_t, \mathbf{a}_t)$ independently and randomly chosen from the distribution $\mathbb{P}(d\mathbf{s}, d\mathbf{a})$. Action $\mathbf{a}_t$ is executed from state $\mathbf{s}_t$ resulting in the system moving to state $\mathbf{s}'_t$. This is recorded along with reward $r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)$ and the action $\mathbf{a}'_t$ that maximizes Q_t when the system is in state $\mathbf{s}'_t$, i.e.,

$$\mathbf{a}'_t := \operatorname*{argmax}_{\mathbf{b}} Q_t(\mathbf{s}'_t, \mathbf{b}). \tag{6.13}$$

The state (S) $\mathbf{s}_t$, action (A) $\mathbf{a}_t$, reward (R) $r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)$, state (S) $\mathbf{s}'_t$, action (A) $\mathbf{a}'_t$ together are called the SARSA tuple. SARSA realizations are required for practical implementation of Q learning, and thus a standard bookkeeping detail.

Further consider the expressions for $J(Q)$ in (6.12) and $L(Q)$ in (6.6) and exchange order of the expectation

and differentiation operators[†] to write the gradient of $J(Q)$ as

$$\nabla_Q J(Q_t) = \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t}\Big[f(Q_t; \mathbf{s}_t, \mathbf{a}_t) \times \nabla_Q f(Q_t; \mathbf{s}_t, \mathbf{a}_t) \Big] + \lambda Q_t \ , \tag{6.14}$$

where $Q \in \mathcal{H}$ and $\nabla_Q J(Q)$ belongs to an RKHS (6.8). To obtain $\nabla_Q f(Q)$ in (6.14), we address differentiation of the softmax and its approximation w.r.t. the exact maximum.

Remark 1 *(Softmax Gradient Error)* The functional derivative of (6.10) takes the form

$$\nabla_Q \text{softmax}_{\mathbf{a}' \in \mathcal{A}} Q(\mathbf{s}', \mathbf{a}') = \frac{\int_{\mathbf{a}' \in \mathcal{A}} e^{\eta Q(\mathbf{s}', \mathbf{a}')} \kappa(\mathbf{s}', \mathbf{a}', \cdot) d\mathbf{a}}{\int_{\mathbf{a}' \in \mathcal{A}} e^{\eta Q(\mathbf{s}', \mathbf{a}')} d\mathbf{a}'} \tag{6.15}$$

by applying Leibniz Rule, Chain Rule, and the reproducing property of the kernel. Moreover, a factor of η cancels. Observe that as $\eta \to \infty$, the softmax becomes closer to the exact (hard) maximum, and the integrals in (6.15) approach unit, and the only term that remains is $\kappa(\mathbf{s}', \mathbf{a}', \cdot)$, where $\mathbf{a}' = \text{argmax}_{\mathbf{b} \in \mathcal{A}} Q(\mathbf{b}, \mathbf{s}')$. This term may be used in place of (6.15) to avoid computing the integral, and yields the functional gradient of the exact maximum instead of the softmax. Doing so comes at the cost of computing of the maximizer of the Q function $\mathbf{a}'$.

Observe that to obtain samples of $\nabla_Q J(Q, \mathbf{s}, \mathbf{a}, \mathbf{s}')$, i.e., the terms inside the outer expectation in (6.14), we require two different queries to a simulation oracle: one to approximate the inner expectation over the Markov transition dynamics defined by $\mathbf{s}'$, and one for each initial pair $\mathbf{s}, \mathbf{a}$ which defines the outer expectation. This complication, called the "double sampling problem," was first identified in [116, 140], has been ameliorated through use of two time-scale stochastic approximation, which may be viewed as a stochastic variant of quasi-gradient methods [125]. We choose to build up the total expectation of one of the terms in (6.14) while doing stochastic descent with respect to the other. In principle, it is possible to build up the expectation of either term in (6.14), but the mean of the difference of kernel evaluations is of infinite complexity. On the other hand, the *temporal action difference*, defined as the difference between the action-value function evaluated at state-action pair $(\mathbf{s}, \mathbf{a})$ and the action-value function evaluated at next state and the instantaneous maximizing action $(\mathbf{s}', \mathbf{a}')$:

$$\delta := r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma Q(\mathbf{s}', \mathbf{a}') - Q(\mathbf{s}, \mathbf{a}) \tag{6.16}$$

[†]This interchange of expectation and differentiation is valid whenever the conditions for the Dominated Convergence Theorem are satisfied. One sufficient condition is that the expression inside the expectation in (6.6) is uniformly bounded. We explicitly ensure this restriction by restricting our Q function estimates in the subsequent section to a compact function space.

is a *scalar*, and thus so is its total expected value. Therefore, for obvious complexity motivations, we build up the total expectation of (6.16). To do so, we propose recursively averaging realizations of (6.16) through the following auxiliary sequence z_t, initialized as null $z_0 = 0$:

$$\delta_t := r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t) + \gamma Q(\mathbf{s}'_t, \mathbf{a}'_t) - Q(\mathbf{s}_t, \mathbf{a}_t) \ , \qquad z_{t+1} = (1 - \beta_t) z_t + \beta_t \delta_t \tag{6.17}$$

where $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t) \overset{i.i.d.}{\sim} (\mathbf{s}, \mathbf{a}, \mathbf{s}')$ and $\beta_t \in (0, 1)$ is a learning rate. In practice, one may obtain i.i.d. samples, but doing so may not yield sufficient visitation to state-action pairs associated with high rewards. Due to this complication, we experimentally investigate deviations from i.i.d. via replay buffers and approximately greedy policies – see Sec. E and Appendix C.

To define the stochastic descent step, we replace the first term inside the outer expectation in (6.14) with its instantaneous approximation $[\gamma\kappa((\mathbf{s}', \mathbf{a}'), \cdot) - \kappa((\mathbf{s}, \mathbf{a}), \cdot)]$ evaluated at a sample triple $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)$, yielding the stochastic quasi-gradient step:

$$Q_{t+1} = (1 - \alpha_t \lambda) Q_t(\cdot) - \alpha_t \left(\gamma\kappa(\mathbf{s}'_t, \mathbf{a}'_t, \cdot) - \kappa(\mathbf{s}_t, \mathbf{a}_t, \cdot) \right) z_{t+1} \tag{6.18}$$

where the coefficient $(1 - \alpha_t \lambda)$ comes from the regularizer and α_t is a positive scalar learning rate. Moreover, $\mathbf{a}'_t = \text{argmax}_{\mathbf{b}} \, Q_t(\mathbf{s}', \mathbf{b})$ is the instantaneous Q-function maximizing action. Now, using similar logic to [124], we may extract a tractable parameterization of the infinite dimensional function sequence (6.18), exploiting properties of the RKHS (6.8).

Kernel Parametrization. Suppose $Q_0 = 0 \in \mathcal{H}$. Then the update (6.18) at time t implies the function Q_t is a kernel expansion of data $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)$

$$Q_t(\mathbf{s}, \mathbf{a}) = \sum_{n=1}^{2(t-1)} w_n \kappa \left(\mathbf{v}_n, (\mathbf{s}, \mathbf{a}) \right) = \mathbf{w}_t^T \kappa_{\mathbf{X_t}} \left((\mathbf{s}, \mathbf{a}) \right) \tag{6.19}$$

The kernel expansion in (6.19), as an instantiation of the Representer Theorem (6.9), together with update (6.18), yields the fact that functional SQG in $\mathcal{H}$ yields updates on the kernel dictionary $\mathbf{X}_t \in \mathbb{R}^{p \times 2(t-1)}$ and coefficients $\mathbf{w}_t \in \mathbb{R}^{2(t-1)}$ as

$$\mathbf{X}_{t+1} = \left[\mathbf{X}_t, (\mathbf{s}_t, \mathbf{a}_t), (\mathbf{s}'_t, \mathbf{a}'_t) \right] \ , \qquad \mathbf{w}_{t+1} = \left[(1 - \alpha_t \lambda) \mathbf{w}_t, \alpha_t z_{t+1}, -\alpha_t \gamma z_{t+1} \right] \tag{6.20}$$

Algorithm 5 Destructive Kernel Orthogonal Matching Pursuit (KOMP)

Input: function $\tilde{Q}$ defined by dict $\tilde{D} \in \mathbb{R}^{p \times \tilde{M}}$, $\tilde{w} \in \mathbb{R}^{\tilde{M}}$, approx. budget $\epsilon_t > 0$
Initialize : $Q = \tilde{Q}$, dictionary $D = \tilde{D}$ with indices $\mathcal{I}$, model order $M = \tilde{M}$, coeffs $w = \tilde{w}$.

```
while candidate dictionary is non-empty I ≠ ∅ do
    for j = 1, ..., M̃ do
        Find minimal approximation error with dictionary element d_j removed
            γ_j = min_{w_{I\{j}} ∈ R^{M-1}} ||Q̃(·) − Σ_{k∈I\{j}} w_k κ(d_k, ·)||_H
    end for
    Find dictionary index minimizing approximation error : j* = argmin_{j∈I} γ_j
    if minimal approximation error exceeds threshold γ_{j*} > ε_t then
        break
    else
        Prune dictionary D ← D_{I\{j*}}
        Revise set I ← I\{j*} and model order M ← M − 1
        Compute updated weights w defined by the current dictionary D
            w = argmin_{w∈R^M} ||Q̃(·) − w^T κ_D(·)||_H
    end if
end while
return V, D, w of model order M ≤ M̃ such that ||Q − Q̃||_H ≤ ε_t
```

In (6.20), the dictionary $\mathbf{X}_t \in \mathbb{R}^{p \times 2(t-1)}$ and the coefficient vector $\mathbf{w}_t \in \mathbb{R}^{2(t-1)}$ are defined as

$$\mathbf{X}_t = \left[(\mathbf{s}_1, \mathbf{a}_1), (\mathbf{s}_1', \mathbf{a}_1'), \ldots, (\mathbf{s}_{t-1}, \mathbf{a}_{t-1}), (\mathbf{s}_{t-1}', \mathbf{a}_{t-1}')\right] \ , \qquad \mathbf{w}_t = \left[w_1, \ldots, w_{2(t-1)}\right] \tag{6.21}$$

and in (6.19), we introduce the notation $\mathbf{v}_n = (\mathbf{s}_n, \mathbf{a}_n)$ for n even and $\mathbf{v}_n = (\mathbf{s}_n', \mathbf{a}_n')$ for n odd. Moreover, in (6.19), define the empirical kernel map associated with dictionary $\mathbf{X}_t$, defined as

$$\kappa_{\mathbf{X}_t}(\cdot) = \left[(\kappa((\mathbf{s}_1, \mathbf{a}_1), \cdot), \kappa((\mathbf{s}_1', \mathbf{a}_1'), \cdot), \ldots, \ldots, \kappa((\mathbf{s}_{t-1}, \mathbf{a}_{t-1}), \cdot), \kappa((\mathbf{s}_{t-1}', \mathbf{a}_{t-1}'), \cdot)\right]^T \ . \tag{6.22}$$

Observe that (6.20) causes $\mathbf{X}_{t+1}$ to have two more columns than its predecessor $\mathbf{X}_t$. Define the *model order* as the number of data points (columns) M_t in the dictionary at time t, which for functional stochastic quasi-gradient descent is $M_t = 2(t-1)$. Asymptotically, the complexity of storing $Q_t(\cdot)$ is infinite, and even for moderately large training sets is untenable. Next, we address this complexity blowup, inspired by [122, 117], using greedy compression [120].

Sparse Stochastic Subspace Projections. Since the update step (6.18) has complexity $\mathcal{O}(t)$ due to the RKHS parametrization, it is impractical in settings with streaming data or arbitrarily large training sets. We address

this issue by replacing the stochastic quasi-descent step (6.18) with an orthogonally projected variant, where the projection is onto a low-dimensional functional subspace of the RKHS $\mathcal{H}_{\mathbf{D}_{t+1}} \subset \mathcal{H}$

$$Q_{t+1} = \mathcal{P}_{\mathcal{H}_{\mathbf{D}_{t+1}}}\left[(1-\alpha_t\lambda)Q_t(\cdot) - \alpha_t(\gamma\kappa(\mathbf{s}_t', \mathbf{a}_t', \cdot) - \kappa(\mathbf{s}_t, \mathbf{a}_t, \cdot))z_{t+1})\right] \tag{6.23}$$

where $\mathcal{H}_{\mathbf{D}_{t+1}} = \text{span}\{((\mathbf{s}_n, \mathbf{a}_n), \cdot)\}_{n=1}^{M_t}$ for some collection of sample instances $\{(\mathbf{s}_n, \mathbf{a}_n)\} \subset \{(\mathbf{s}_t, \mathbf{a}_t)\}_{u \leq t}$. We define $\kappa_{\mathbf{D}}(\cdot) = \{\kappa((\mathbf{s}_1, \mathbf{a}_1), \cdot) \ldots \kappa((\mathbf{s}_M, \mathbf{a}_M), \cdot)\}$ and $\kappa_{\mathbf{D},\mathbf{D}}$ as the resulting kernel matrix from this dictionary. We seek function parsimony by selecting dictionaries $\mathbf{D}$ such that $M_t << \mathcal{O}(t)$. Suppose that Q_t is parameterized by model points $\mathbf{D}_t$ and weights $\mathbf{w}_t$. Then, we denote $\tilde{Q}_{t+1}(\cdot) = (1-\alpha_t\lambda)Q_t(\cdot) - \alpha_t(\gamma\kappa(\mathbf{s}_t', \mathbf{a}_t', \cdot) - \kappa(\mathbf{s}_t, \mathbf{a}_t, \cdot))z_{t+1})$ as the SQG step without projection. This may be represented by dictionary and weight vector [cf. (6.20)]:

$$\tilde{\mathbf{D}}_{t+1} = \left[\mathbf{D}_t, (\mathbf{s}_t, \mathbf{a}_t), (\mathbf{s}_t', \mathbf{a}_t')\right] \ , \quad \tilde{\mathbf{w}}_{t+1} = [(1-\alpha_t\lambda)\mathbf{w}_t, \alpha_t z_{t+1}, -\alpha_t\gamma z_{t+1}] \ , \tag{6.24}$$

where z_{t+1} in (6.24) is computed by (6.17) using Q_t in (6.23):

$$\delta_t := r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_t') + \gamma Q_t(\mathbf{s}_t', \mathbf{a}_t') - Q_t(\mathbf{s}_t, \mathbf{a}_t) \ , \quad z_{t+1} = (1-\beta_t)z_t + \beta_t\delta_t \ . \tag{6.25}$$

Observe that $\tilde{\mathbf{D}}_{t+1}$ has $\tilde{M}_{t+1} = M_t + 2$ columns which is the length of $\tilde{\mathbf{w}}_{t+1}$. We proceed to describe the construction of the subspaces $\mathcal{H}_{\mathbf{D}_{t+1}}$ onto which the SQG iterates are projected in (6.23). Specifically, we select the kernel dictionary $\mathbf{D}_{t+1}$ via destructive subset selection: we form $\mathbf{D}_{t+1}$ starting with the full dictionary and sequentially removing elements. Specifically, we select a subset of M_{t+1} columns from $\tilde{\mathbf{D}}_{t+1}$ that best approximates $\tilde{Q}_{t+1}$ in terms of Hilbert norm error. To accomplish this, we use kernel orthogonal matching pursuit [122, 117] with error ϵ_t to find a compressed dictionary $\mathbf{D}_{t+1}$ from $\tilde{\mathbf{D}}_{t+1}$, the one that adds the latest samples. For fixed dictionary $\mathbf{D}_{t+1}$, the coefficient update is a least-squares multiplication:

$$\mathbf{w}_{t+1} = \kappa_{\mathbf{D}_{t+1}\mathbf{D}_{t+1}}^{-1}\kappa_{\mathbf{D}_{t+1}\tilde{\mathbf{D}}_{t+1}}\tilde{\mathbf{w}}_{t+1} \tag{6.26}$$

We tune ϵ_t to ensure stochastic descent and finite model order. Due to the need to guarantee Assumption 1, i.e., ensuring $\|Q_t\|$ is bounded, we further require that (6.26) the coefficient vector is intersected with a finite-norm ball, i.e., very large values are thresholded by a constant. Since in practice this thresholding is omitted and

only used for ensuring boundedness, we do not comment further on this point outside of Assumption 1 and (6.38).

We summarize the proposed method, KQ-Learning, in Algorithm 6, the execution of the stochastic projection of the functional SQG iterates onto subspaces $\mathcal{H}_{\mathbf{D}_{t+1}}$. We begin with a null function $Q_0 = 0$, i.e., empty dictionary and coefficients (Step 1). At each step, given an i.i.d. sample $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)$ and step-size α_t, β_t (Steps 2-5), we compute the unconstrained functional SQG iterate $\tilde{Q}_{t+1}(\cdot) = (1-\alpha_t\lambda)Q_t(\cdot) - \alpha_t(\gamma\kappa(\mathbf{s}'_t, \mathbf{a}'_t, \cdot) - \kappa(\mathbf{s}_t, \mathbf{a}_t, \cdot))z_{t+1})$ parametrized by $\tilde{\mathbf{D}}_{t+1}$ and $\tilde{\mathbf{w}}_{t+1}$ (Steps 6-7), which are fed into KOMP (Algorithm 5) [122] with budget ϵ_t, (Step 8). KOMP then returns a lower complexity estimate Q_t of $\tilde{Q}_t$ that is ϵ_t away in $\mathcal{H}$. The complexity control properties of Algorithm 5 as a projection are illuminated next.

Remark 2 *In [122][Section 4.3], the complexity of Algorithm 5 is characterized as depending on the model order M_t and the number of removed points Z_t as $\mathcal{O}(M_t^2 Z_t)$. Intuitively, this makes projections more costly to compute when ϵ is smaller, as this makes the model complexity increase.*

That is, for constant step-size and compression parameter, Corollary 1 of [117] applies, which, together with [141][Proposition 2.2], implies that $\max_t M_t \leq Y[\frac{\beta\epsilon}{\alpha}]^l$. *$Y$ is a constant that depends on kernel hyper-parameters (e.g., Gaussian kernel bandwidth) and smoothness parameters of the objective w.r.t. the Q function, and $l = p + q$ is the parameter dimension.*

By substituting this approximate expression into the aforementioned complexity analysis, we may obtain an estimate of the effort required to compute Algorithm 5 as $\mathcal{O}((\frac{\alpha}{\epsilon})^{2l} Z_t)$. This indeed illuminates that for small ϵ, the projection becomes intractable, and confirms the motivation for holding constant both step-size and compression budget. The performance tradeoffs of this selection are formalized in the subsequent section.

C Practical Considerations

The convergence guarantees for Algorithm 6 require sequentially observing state-action-next-state triples $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}'_t)$ independently and identically distributed. Doing so, however, only yields convergence toward a stationary point, which may or may not be the optimal Q function. To improve the quality of stationary points to which we converge, it is vital to observe states that yield reward (an instantiation of the explore-exploit tradeoff). To do so, we adopt a standard practice in reinforcement learning which is to bias actions towards states that may accumulate more reward.

Algorithm 6 KQ-Learning

Input: $C, \{\alpha_t, \beta_t\}_{t=0,1,2\ldots}, Q_0(\cdot) = 0, D_0 = [], w_0 = [], z_0 = 0$

for $t = 0, 1, 2, \ldots$ **do**
  Obtain sample $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)$ via exploratory policy
  Compute maximizing action, $\mathbf{a}'_t$ (6.13)
  Update temporal action diff. δ_t and aux. seq. z_{t+1} (6.25)
  Compute functional stochastic quasi-grad. step

$$\tilde{Q}_{t+1} = (1 - \alpha_t \lambda) Q_t - \alpha_t z_{t+1} \left(\gamma \kappa(\mathbf{s}'_t, \mathbf{a}'_t, \cdot) - \kappa(\mathbf{s}_t, \mathbf{a}_t, \cdot) \right).$$

  Update dictionary and weights (6.24)
  Compress function using KOMP with budget $\epsilon_t = C\alpha_t^2$

$$(Q_{t+1}, D_{t+1}, \mathbf{w}_{t+1}) = \textbf{KOMP}(\tilde{Q}_{t+1}, \tilde{D}_{t+1}, \tilde{w}_{t+1}, \epsilon_t)$$

end for
return Q

Algorithm 7 Semi-Gradient Greedy KQ-Learning

Input: $C, \{\alpha_t, \beta_t\}_{t=0,1,2\ldots}$

$Q_0(\cdot) = 0, D_0 = [], w_0 = [], z_0 = 0$
for $t = 0, 1, 2, \ldots$ **do**
  Obtain sample $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)$ via greedy policy
  Compute maximizing action:

$$\mathbf{a}'_t = \pi_t(\mathbf{s}'_t) = \text{argmax}_{\mathbf{a}} \, Q_t(\mathbf{s}'_t, \mathbf{a})$$

  Update temporal action diff. δ_t and aux. seq. z_{t+1}

$$\delta_t = r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t) + \gamma Q_t(\mathbf{s}'_t, \mathbf{a}'_t) - Q_t(\mathbf{s}_t, \mathbf{a}_t), \quad z_{t+1} = (1 - \beta_t) z_t + \beta_t \delta_t$$

  Compute update step

$$\tilde{Q}_{t+1} = (1 - \alpha_t \lambda) Q_t(\cdot) + \alpha_t z_{t+1} \kappa(\mathbf{s}_t, \mathbf{a}_t, \cdot)$$

  Update dictionary $\tilde{D}_{t+1} = [D_t, (\mathbf{s}, \mathbf{a})]$ and weights $\tilde{w}_{t+1} = [(1 - \alpha_t \lambda) w_t, \alpha_t z_{t+1}]$
  Compress with budget $\epsilon_t = C\alpha_t^2$: $(Q_{t+1}, D_{t+1}, w_{t+1}) = \textbf{KOMP}(\tilde{Q}_{t+1}, \tilde{D}_{t+1}, \tilde{w}_{t+1}, \epsilon_t)$
end for
return Q

The method in which we propose to bias actions is by selecting them according to the current estimate of the optimal policy, i.e., the greedy policy. However, when doing so, the KQ-Learning updates (6.18) computed using greedy samples $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}'_t)$ are composed of two points nearby in $\mathcal{S} \times \mathcal{A}$ space. These points are then evaluated by kernels and given approximately equal in opposite weight. Thus, this update is immediately pruned away during the execution of KOMP [120, 142, 122]. In order to make use of the greedy samples and speed up convergence, we project the functional update onto just one kernel dictionary element, resulting in the update step:

$$\tilde{Q}_{t+1} = (1 - \alpha_t \lambda) Q_t(\cdot) + \alpha_t \left(\kappa(\mathbf{s}_t, \mathbf{a}_t, \cdot) \right) z_{t+1} \tag{6.27}$$

The resulting procedure is summarized as Algorithm 7. First, trajectory samples are obtained using a greedy policy. Then, the temporal-action difference is computed and averaged recursively. Finally, we update Q via (6.27) and compress it using Algorithm 5.

ρ-Greedy Actions and Hybrid Update. To address the explore-exploit trade-off, we use an ρ-greedy policy [143]: with probability ρ we select a random action, and select a greedy action with probability $1 - \rho$. We adopt this approach with ρ decreasing linearly during training, meaning that as time passes more greedy actions are taken.

The algorithm when run with a ρ-greedy policy is described as the *Hybrid* algorithm, which uses Algorithm 6 when exploratory actions are taken and Algorithm 7 for greedy actions. Practically, we find it useful to judiciously use training examples, which may be done with a data buffer. Thus, the hybrid algorithm is as follows: First, we accumulate trajectory samples in a buffer. Along with the $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}'_t)$ sample, we store an indicator whether $\mathbf{a}_t$ was an exploratory action or greedy with respect to Q_{t-1}. Then, samples are drawn at random from the buffer for training. We explore two different methods for obtaining samples from the buffer: uniformly at random, and prioritized sampling, which weighs each sample in the buffer by its observed Bellman error. For greedy actions, we use the update in (6.27), and for exploratory actions, we use the KQ-learning update from 6. Finally, we use KOMP to compress the representation of the Q function.

Maximizing the Q Function. In order to implement Algorithm 7, we must evaluate the instantaneous maximizing action $\mathbf{a}_t = \operatorname{argmax}_{\mathbf{a}} Q_t((\mathbf{s}_t, \mathbf{a}))$. For a general reproducing kernel $\kappa(\cdot, \cdot)$, maximizing over a weighted sum of kernels is a non-convex optimization problem, so we may get stuck in undesirable stationary points. To reduce the chance that this undesirable outcome transpires, we use simulated annealing. First, we sample actions $\mathbf{a}$ uniformly at random from the action space. Next, we use gradient ascent to refine our estimate of the global maximum of Q for state $\mathbf{s}$. We use the Gaussian Radial Basis Function(RBF) kernel (6.11), so the Q function is differentiable with respect to an arbitrary action $\mathbf{a}$:

$$(\nabla_{\mathbf{a}} Q)(\mathbf{s}, \mathbf{a}) = Q(\mathbf{s}, \mathbf{a}) \sum_{m=1}^{M} w_m \Sigma_{\mathbf{a}} (\mathbf{a} - \mathbf{a}_m)^T \tag{6.28}$$

and that gradient evaluations are cheap: typically their complexity scales with the model order of the Q function which is a kept under control using Algorithm 5.

Remark 3 Observe that (6.27) bears a phantom resemblance to Watkins' Q-Learning algorithm [105]; how-

ever, it is unclear how to extend [105] to continuous MDPs where function approximation is required. In practice, using (6.27) for all updates, we observe globally steady policy learning and convergence of Bellman error, suggesting a link between (6.27) and stochastic fixed point methods [109, 110]. This link is left to future investigation. For now, we simply note that stochastic fixed point iteration is fundamentally different than stochastic descent methods which rely on the construction of supermartingales, so results from the previous section do not apply to (6.27). Moreover, this update has also been referred to as a temporal difference "semi-gradient" update in Chapters 9-10 of [101].

D Convergence Analysis

In this section, we shift focus to the task of establishing that the sequence of action-value function estimates generated by Algorithm 6 yields a stationary solution to the Bellman optimality equation, which, given intrinsic the non-convexity of the problem setting, is the best one may hope for in general through use of numerical stochastic optimization methods. Our analysis extends the ideas of coupled supermartingales in reproducing kernel Hilbert spaces [117], which have been used to establish convergent policy evaluation approaches in infinite MDPs (a convex problem), to non-convex settings, and further generalizes the non-convex vector-valued setting of [125].

Under the technical conditions stated to be subsequently stated, we derive a coupled stochastic descent-type relationship for Algorithm 6 and then apply the Coupled Supermartingale Theorem [144][Lemma 6] to establish convergence. Before proceeding, we introduce a few definitions which simplify derivations greatly. In particular, for further reference, we use (6.13) to define $\mathbf{a}'_t = \text{argmax}_{\mathbf{a}} \, Q_t(\mathbf{s}'_t, \mathbf{a})$, the instantaneous maximizer of the action-value function and defines the direction of the gradient. We also define the functional stochastic quasi-gradient of the regularized objective

$$\hat{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t) = \left(\gamma \kappa \left((\mathbf{s}'_t, \mathbf{a}'_t), \cdot\right) - \kappa \left((\mathbf{s}_t, \mathbf{a}_t), \cdot\right)\right) z_{t+1} + \lambda Q_t \tag{6.29}$$

and its sparse-subspace projected variant as

$$\tilde{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t) = \left(Q_t - \mathcal{P}_{\mathcal{H}_{D_{t+1}}} \left[Q_t - \alpha_t \hat{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)\right]\right) / \alpha_t \tag{6.30}$$

Note that the update may be rewritten as a stochastic projected quasi-gradient step rather than a stochastic quasi-gradient step followed by a set projection, i.e.,

$$Q_{t+1} = Q_t - \alpha_t \tilde{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t) \tag{6.31}$$

Further denote as the filtration $\mathcal{F}_t$ as the time-dependent sigma-algebra containing the algorithm history $(\{Q_u, z_u\}_{u=0}^t \cup \{\mathbf{s}_u, \mathbf{a}_u, \mathbf{s}'_u\}_{u=0}^{t-1}) \subset \mathcal{F}_t$. With these definitions, we may state our main assumptions required to establish convergence of Algorithm 6.

Assumption 1 *The state space $\mathcal{S} \subset \mathbb{R}^p$ and action space $\mathcal{A} \subset \mathbb{R}^q$ are compact, and the reproducing kernel map may be bounded with probability 1 as*

$$\sup_{\mathbf{s}\in\mathcal{S}, \mathbf{a}\in\mathcal{A}} \sqrt{\kappa((\mathbf{s}, \mathbf{a}), (\mathbf{s}, \mathbf{a}))} = K < \infty \tag{6.32}$$

Moreover, the subspaces $\mathcal{H}_{\mathbf{D}_t}$ are intersected with some finite Hilbert norm ball for each t.

Assumption 2 *The temporal action difference δ and auxiliary sequence z satisfy the zero-mean, finite conditional variance, and Lipschitz continuity conditions, respectively,*

$$\mathbb{E}[\delta \mid \mathbf{s}, \mathbf{a}] = \bar{\delta}\,, \quad \mathbb{E}[(\delta - \bar{\delta})^2 \,\big|\, \mathbf{s}, \mathbf{a}, \mathbf{s}'] \leq \sigma_\delta^2\,, \quad \mathbb{E}[z^2 \mid \mathbf{s}, \mathbf{a}] \leq G_\delta^2 \;\; \textit{for all } Q \in \mathcal{H} \tag{6.33}$$

where σ_δ and G_δ are positive scalars, and $\bar{\delta} = \mathbb{E}\{\delta \mid \mathbf{s}, \mathbf{a}\}$ is the expected value of the temporal action difference conditioned on the state $\mathbf{s}$ and action $\mathbf{a}$. The expectation defining σ_δ^2 is conditional on $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$. These conditions hold with probability 1 for all $\mathbf{s}, \mathbf{a}, \mathbf{s}'$.

Assumption 3 *The functional gradient of the temporal difference is an unbiased estimate for $\nabla_Q J(Q)$ and the difference of the reproducing kernels expression has finite variance:*

$$\mathbb{E}\left[(\gamma\kappa((\mathbf{s}', \mathbf{a}'), \cdot) - \kappa((\mathbf{s}, \mathbf{a}), \cdot))\bar{\delta}\right] = \nabla_Q J(Q) \tag{6.34}$$

$$\mathbb{E}\left\{\|\gamma\kappa((\mathbf{s}', \mathbf{a}'), \cdot) - \kappa((\mathbf{s}, \mathbf{a}), \cdot)\|_{\mathcal{H}}^2 \,\big|\, \mathbf{s}, \mathbf{a}, \mathbf{s}'\right\} \leq G_Q^2 \tag{6.35}$$

Moreover, the projected stochastic quasi-gradient of the objective has finite second moment

$$\mathbb{E}\left\{\|\tilde{\nabla}_Q J(Q, z; \mathbf{s}, \mathbf{a}, \mathbf{s}')\|_{\mathcal{H}}^2 \,\middle|\, \mathbf{s}, \mathbf{a}, \mathbf{s}'\right\} \leq \sigma_Q^2 \tag{6.36}$$

and the temporal action difference is Lipschitz continuous with respect to the action-value function Q. Moreover, for any two distinct $\bar{\delta}_u = \mathbb{E}[\delta_u \mid \mathbf{s}_u, \mathbf{a}_u]$ and $\bar{\delta}_t = \mathbb{E}[\delta_t \mid \mathbf{s}_t, \mathbf{a}_t]$, we have

$$\|\bar{\delta}_u - \bar{\delta}_t\| \leq L_Q \|Q_u - Q_t\|_{\mathcal{H}} \tag{6.37}$$

with Q_u, $Q_t \in \mathcal{H}$ distinct Q-functions; $L_Q > 0$ is a scalar. The aforementioned conditions hold with probability 1 *for all* $\mathbf{s}, \mathbf{a}, \mathbf{s}'$.

Assumption 1 regarding the compactness of the state and action spaces of the MDP holds for most application settings and limits the radius of the set from which the MDP trajectory is sampled. The mean and variance properties of the temporal difference stated in Assumption 2 are necessary to bound the error in the descent direction associated with the stochastic sub-sampling and are required to establish convergence of stochastic methods. Regarding the last statement in Assumption 2, observe that z_t is a recursive average of δ_t, which is comprised of bounded terms (finite-norm Q-functions and a finite reward). Therefore, it naturally holds that z_t is an estimator whose sample path variance is finite. The finite variance of δ is also reasonable due to the fact that the reward and Q functions are bounded. Assumption 3 is similar to Assumption 2, but instead of establishing bounds on the stochastic approximation error of the temporal difference, limits stochastic error variance in the RKHS. The term related to the maximum of the Q function in the temporal action difference is Lipschitz in the infinity norm since Q is automatically Lipschitz since it belongs to the RKHS. Thus, this term can be related to the Hilbert norm through a constant factor. Hence, (6.37) is only limits how non-smooth the reward function may be. These are natural extensions of the conditions needed for vector-valued stochastic compositional gradient methods.

Due to Assumption 1 and the use of set projections in (6.23), we have that Q_t is always bounded in Hilbert norm, i.e., there exists some $0 < D < \infty$ such that

$$\|Q_t\|_{\mathcal{H}} \leq D \text{ for all } t \ . \tag{6.38}$$

Under these conditions, we may establish a few key technical results which are precursors to our main convergence results. In particular, we have the following proposition that characterizes the directional error caused by projections. (6.38) may be explicitly enforced by thresholding the weight vectors w_t to have norm smaller than a large constant, and therefore this is a restriction on the range of the weight vectors output by the algorithm, rather than an assumption on the problem class itself.

Proposition 1 *Given independent identical realizations* $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)$ *of the random triple* $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$*, the difference between the projected stochastic functional quasi-gradient and the stochastic functional quasi-gradient of the instantaneous cost is bounded for all* t *as*

$$\|\tilde{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t) - \hat{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t\, \mathbf{a}_t, \mathbf{s}'_t)\|_{\mathcal{H}} \leq \frac{\epsilon_t}{\alpha_t} \tag{6.39}$$

Where $\alpha_t > 0$ *denotes the step size and* $\epsilon_t > 0$ *is the compression parameter of KOMP.*

Proof 1 *See Appendix F.*

Lemma 1 *Denote the filtration* $\mathcal{F}_t$ *as the time-dependent sigma-algebra containing the algorithm history. Let Assumptions 1-3 hold true and consider the sequence of iterates defined by Algorithm 6. Then:*

1. *The conditional expectation of the Hilbert-norm difference of action-value functions at the next and current iteration satisfies the relationship*

$$\mathbb{E}\left[\|Q_{t+1} - Q_t\|^2_{\mathcal{H}} \mid \mathcal{F}_t\right] \leq 2\alpha_t^2 \left(G_\delta^2 G_Q^2 + \lambda D^2\right) + 2\epsilon_t^2 \tag{6.40}$$

2. *The auxiliary sequence* z_t *with respect to the conditional expectation of the temporal action difference* $\bar{\delta}_t$ *(defined in Assumption 2) satisfies*

$$\mathbb{E}\left[(z_{t+1} - \bar{\delta}_t)^2 \mid \mathcal{F}_t\right] \leq (1 - \beta_t)(z_t - \bar{\delta}_{t-1})^2 + \frac{L_Q}{\beta_t}\|Q_t - Q_{t-1}\|^2_{\mathcal{H}} + 2\beta_t^2\sigma_\delta^2 \tag{6.41}$$

3. *Algorithm 6 generates a sequence of Q-functions that satisfy the stochastic descent property with respect*

to the Bellman error $J(Q)$ [cf. (6.12)*]:*

$$\mathbb{E}\left[J(Q_{t+1})\,\middle|\,\mathcal{F}_t\right] \leq J(Q_t) - \alpha_t\Big(1 - \frac{\alpha_t G_Q^2}{\beta_t}\Big)\|\nabla_Q J(Q)\|^2 + \frac{\beta_t}{2}\mathbb{E}\left[(\bar{\delta}_t - z_{t+1})^2 \mid \mathcal{F}_t\right] + \frac{L_Q \sigma_Q^2 \alpha_t^2}{2} + \epsilon_t\|\nabla_Q J(Q_t)\|_{\mathcal{H}}\ , \tag{6.42}$$

Proof 2 *See Appendix G.*

With these technical results regarding the per-step behavior of Algorithm 6 established, we are ready to present our main convergence results. We begin by discussing algorithm stability under attenuating step-size rules.

Theorem 1 *Consider the sequence z_t and $\{Q_t\}$ as stated in Algorithm 6. Assume the regularizer is positive $\lambda > 0$, Assumptions 1-3 hold, and the step-size conditions hold, with $C > 0$ a positive constant:*

$$\sum_{t=1}^{\infty}\alpha_t = \infty, \sum_{t=1}^{\infty}\beta_t = \infty, \sum_{t=1}^{\infty}\alpha_t^2 + \beta_t^2 + \frac{\alpha_t^2}{\beta_t} < \infty, \epsilon_t = C\alpha_t^2 \tag{6.43}$$

Then $\|\nabla_Q J(Q)\|_{\mathcal{H}}$ converges to null with probability 1, and hence Q_t attains a stationary point of (6.12)*. In particular, the limit of Q_t achieves the regularized Bellman fixed point restricted to the RKHS.*

Proof: We use the relations established in Lemma 1 to construct a coupled supermatingale of the form 2. First, we state the following lemma regarding coupled sequences of conditionally decreasing stochastic processes called the coupled supermartingale lemma, stated as:

Lemma 2 *(Coupled Supermartingale Theorem) [144]. Let $\{\xi_t\}$, $\{\zeta_t\}$, $\{u_t\}$, $\{\bar{u}_t\}$, $\{\eta_t\}$, $\{\theta_t\}$, $\{\epsilon_t\}$, $\{\mu_t\}$, $\{\nu_t\}$ be sequences of nonnegative random variables such that*

$$\begin{aligned}\mathbb{E}\{\xi_{t+1} \mid G_t\} &\leq (1+\eta_t)\xi_t - u_t + c\theta_t\zeta_t + \mu_t\ ,\\ \mathbb{E}\{\zeta_{t+1} \mid G_t\} &\leq (1-\theta_t)\zeta_t - \bar{u}_t + \epsilon_t\xi_t + \nu_t\end{aligned} \tag{6.44}$$

where $G_t = \{\xi_s, \xi_s, u_s, \bar{u}_s, \eta_s, \theta_s, \epsilon_s, \mu_s, \nu_s\}_{s=0}^{t}$ is the filtration and $c > 0$ is a scalar. Suppose the following

summability conditions hold almost surely:

$$\sum_{t=0}^{\infty} \eta_t < \infty \, , \sum_{t=0}^{\infty} \epsilon_t < \infty \, , \sum_{t=0}^{\infty} \mu_t < \infty \, , \sum_{t=0}^{\infty} \nu_t < \infty \tag{6.45}$$

Then ξ_t and ζ_t converge almost surely to two respective nonnegative random variables, and we may conclude that almost surely

$$\sum_{t=0}^{\infty} u_t < \infty \, , \sum_{t=0}^{\infty} \bar{u}_t < \infty \, , \sum_{t=0}^{\infty} \theta_t \zeta_t < \infty \tag{6.46}$$

We construct coupled supermartingales that match the form of Lemma 2 using Lemma 1. First, use Lemma 1(2)(6.41) as an upper bound on Lemma 1(3) (6.42).

$$\begin{aligned} \mathbb{E}[J(Q_{t+1}) \,\big|\, \mathcal{F}_t] \leq J(Q_t) - \alpha_t (1 - \frac{\alpha_t G_Q^2}{\beta_t}) \|\nabla_Q J(Q_t)\|^2 + \frac{\beta_t}{2}((1-\beta_t)(z_t - \bar{\delta}_{t-1})^2 \\ + \frac{L_Q}{\beta_t}\|Q_t - Q_{t-1}\|_{\mathcal{H}}^2 + 2\beta_t^2 \sigma_\delta^2) + \frac{L_Q \sigma_Q^2 \alpha_t^2}{2} + \epsilon_t \|\nabla_Q J(Q_t)\|_{\mathcal{H}} \end{aligned} \tag{6.47}$$

We introduce three restrictions on the learning rate, expectation rate, and parsimony constant in order to simplify (6.47). First, we assume that $\beta_t \in (0,1)$ for all t. Next, we choose $\epsilon_t = \alpha_t^2$. Lastly, we restrict $1 - \frac{\alpha_t G_Q^2}{\beta_t} > 0$, which results in the condition: $\frac{\alpha_t}{\beta_t} < \frac{1}{G_Q^2}$. Then, we simplify and group terms of (6.47).

$$\begin{aligned} \mathbb{E}\left[J(Q_{t+1}) \,\big|\, \mathcal{F}_t\right] \leq J(Q_t) - \alpha_t \|\nabla_Q J(Q_t)\|^2 + \frac{\beta_t}{2}(z_t - \bar{\delta}_{t-1})^2 \\ + \frac{L_Q}{2}\|Q_t - Q_{t-1}\|_{\mathcal{H}}^2 + \beta_t^3 \sigma_\delta^2 + \alpha_t^2 \left(\frac{L_Q \sigma_Q^2}{2} + \|\nabla_Q J(Q_t)\|_{\mathcal{H}} \right) \end{aligned} \tag{6.48}$$

Next, we aim to connect the result of (6.47) to the form of Lemma 2 via the identifications:

$$\begin{aligned} &\xi_t = J(Q_t) \, , \zeta_t = (z_t - \bar{\delta}_{t-1})^2 \, , \theta_t = \beta_t \, , c = 1/2 \, , u_t = \alpha_t \|\nabla_Q J(Q_t)\|^2 \, , \eta_t = 0 \\ &\mu_t = \frac{L_Q}{2}\|Q_t - Q_{t-1}\|_{\mathcal{H}}^2 + \beta_t^3 \sigma_\delta^2 + \alpha_t^2 \left(\frac{L_Q \sigma_Q^2}{2} + \|\nabla_Q J(Q_t)\|_{\mathcal{H}} \right) \end{aligned} \tag{6.49}$$

Observe that $\sum \mu_t < \infty$ due to the upper bound on $\|Q_t - Q_{t-1}\|_{\mathcal{H}}^2$ provided by Lemma 1(6.40) and the summability conditions for α_t^2 and β_t^2 (6.43).

Next, we identify terms in Lemma 1 (2) (6.41) according to Lemma 2 and (6.49).

$$\nu_t = \frac{L_Q}{\beta_t}\|Q_t - Q_{t-1}\|^2_{\mathcal{H}} + 2\beta_t^2\sigma_\delta^2 \ , \quad \epsilon_t = 0 \ , \quad \bar{u}_t = 0 \tag{6.50}$$

The summability of ν_t can be shown as follows: the expression $\|Q_t - Q_{t-1}\|^2_{\mathcal{H}}/\beta_t$ which is order $\mathcal{O}(\alpha_t^2/\beta_t)$ in conditional expectation by Lemma 1(1). Sum the resulting conditional expectation for all t, which by the summability of the sequence $\sum_t \alpha_t^2/\beta_t < \infty$ is finite. Therefore, $\sum_t \|Q_t - Q_{t-1}\|^2_{\mathcal{H}}/\beta_t < \infty$ almost surely. We also require $\sum_t \beta_t^2 < \infty$ (6.43) for the summability of the second term of (6.50)

Together with the conditions on the step-size sequences α_t and β_t, the summability conditions of Lemma 2 are satisfied, which allows to conclude that $\xi_t = J(Q_t)$ and $\zeta_t = (z_t - \bar{\delta}_{t-1})^2$ converge to two nonnegative random variables w.p. 1, and that

$$\sum_t \alpha_t \|\nabla_Q J(Q_t)\|^2 < \infty, \quad \sum_t \beta_t (z_t - \bar{\delta}_{t-1})^2 < \infty \tag{6.51}$$

almost surely. Then, the summability of u_t taken together with non-summability of α_t and β_t (6.43) indicates that the limit infimum of the norm of the gradient of the cost goes to null.

$$\liminf_{t\to\infty} \|\nabla_Q J(Q_t)\| = 0 \ , \quad \liminf_{t\to\infty} (z_t - \bar{\delta}_{t-1})^2 = 0 \tag{6.52}$$

almost surely. From here, given $\liminf_{t\to\infty} \|\nabla_Q J(Q_t)\|_{\mathcal{H}} = 0$, we can apply almost the exact same argument by contradiction as [125] to conclude that the whole sequence $\|\nabla_Q J(Q_t)\|_{\mathcal{H}}$ converges to null with probability 1, which is repeated here for completeness.

Consider some $\eta > 0$ and observe that $\|\nabla_Q J(Q_t)\|_{\mathcal{H}} \leq \eta$ for infinitely many t. Else, there exists t_0 such that $\sum_{t=0}^{\infty} \|\alpha_t \nabla_Q J(Q_t)\|^2_{\mathcal{H}} \geq \sum_{t=t_0}^{\infty} \alpha_t \eta^2 = \infty$ which contradicts (6.51). Therefore, there exists a closed set $\bar{\mathcal{H}} \subset \mathcal{H}$ such that $\{Q_t\}$ visits $\bar{\mathcal{H}}$ infinitely often, and

$$\|\nabla_Q J(Q)\|_{\mathcal{H}} \begin{cases} \leq \eta \text{ for } Q \in \bar{\mathcal{H}} \\ > \eta \text{ for } Q \notin \bar{\mathcal{H}}, Q \in \{Q_t\} \end{cases} \tag{6.53}$$

Suppose to the contrary that there exists a limit point $\tilde{Q}$ such that $\|\nabla_Q J(\tilde{Q})\|_{\mathcal{H}} > 2\eta$. Then there exists a

closed set $\tilde{\mathcal{H}}$, i.e., a union of neighborhoods of all Q_t's such that $\|\nabla_Q J(Q_t)\|_{\mathcal{H}} > 2\eta$, with $\{Q_t\}$ visiting $\tilde{\mathcal{H}}$ infinitely often, and

$$\|\nabla_Q J(Q)\|_{\mathcal{H}} \begin{cases} \geq 2\eta \text{ for } Q \in \tilde{\mathcal{H}} \\ < 2\eta \text{ for } Q \notin \tilde{\mathcal{H}}, Q \in \{Q_t\} \end{cases} \tag{6.54}$$

Using the continuity of ∇J and $\eta > 0$, we have that $\bar{\mathcal{H}}$ and $\tilde{\mathcal{H}}$ are disjoint: $\text{dist}(\bar{\mathcal{H}}, \tilde{\mathcal{H}}) > 0$. Since $\{Q_t\}$ enters both $\bar{\mathcal{H}}$ and $\tilde{\mathcal{H}}$ infinitely often, there exists a subsequence $\{Q_t\}_{t\in\mathcal{T}} = \{\{Q_t\}_{t=k_i}^{j_i-1}\}$ (with $\mathcal{T} \subset \mathbb{Z}^+$) that enters $\bar{\mathcal{H}}$ and $\tilde{\mathcal{H}}$ infinitely often, with $Q_{k_i} \in \bar{\mathcal{H}}$ and $Q_{j_i} \in \tilde{\mathcal{H}}$ for all i. Therefore, for all i, we have

$$\|\nabla_Q J(Q_{k_i})\|_{\mathcal{H}} \geq 2\eta > \|\nabla_Q J(Q_t)\|_{\mathcal{H}} > \eta \geq \|\nabla_Q J(Q_{j_i})\|_{\mathcal{H}} \quad \text{for } t = k_i + 1, \ldots, j_i - 1$$

Therefore, we can write

$$\sum_{t\in\mathcal{T}} \|Q_{t+1} - Q_t\|_{\mathcal{H}} = \sum_{i=1}^{\infty} \sum_{t=k_i}^{j_i-1} \|Q_{t+1} - Q_t\|_{\mathcal{H}} \geq \sum_{i=1}^{\infty} \|Q_{k_i} - Q_{j_i}\|_{\mathcal{H}} \geq \text{dist}(\bar{\mathcal{H}}, \tilde{\mathcal{H}}) = \infty \tag{6.55}$$

However, we may also write that

$$\infty > \sum_{t=0}^{\infty} \alpha_t \|\nabla J(Q_t)\|_{\mathcal{H}}^2 \geq \sum_{t\in\mathcal{T}} \alpha_t \|\nabla J(Q_t)\|_{\mathcal{H}}^2 > \eta^2 \sum_{t\in\mathcal{T}} \alpha_t \tag{6.56}$$

Then, since sets $\mathcal{X}$ and $\mathcal{A}$ are compact, there exist some $M > 0$ such that

$$\|Q_{t+1} - Q_t\|_{\mathcal{H}} \leq \alpha_t \|\tilde{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_t')\|_{\mathcal{H}} \leq M\alpha_t \tag{6.57}$$

for all t, using the fact that $\epsilon_t = \alpha_t^2$. Therefore,

$$\sum_{t\in\mathcal{T}} \|Q_{t+1} - Q_t\|_{\mathcal{H}} \leq M \sum_{t\in\mathcal{T}} \alpha_t < \infty \tag{6.58}$$

which contradicts (6.55). Therefore, there does not exist any limit point $\tilde{Q}$ such that $\|\nabla_Q J(\tilde{Q})\|_{\mathcal{H}} > 2\eta$. By making η arbitrarily small, it means that there does not exist any limit point that is nonstationary. Moreover, we note that the set of such sample paths occurs with probability 1, since the preceding analysis applies to all

sample paths which satisfy (6.51). Thus, any limit point of Q_t is a stationary point of $J(Q)$ almost surely. ■

Theorem 1 establishes that Algorithm 6 converges almost surely to a stationary solution of the problem (6.12) defined by the Bellman optimality equation in a continuous MDP. This is one of the first Lyapunov stability results for Q-learning in continuous state-action spaces with nonlinear function parameterizations, which are intrinsically necessary when the Q-function does not admit a lookup table (matrix) representation, and should form the foundation for value-function based reinforcement learning in continuous spaces. One way to satisfy the step-size conditions in (6.43) is to set $\alpha_t = t^{-a}$, $\beta_t = t^{-b}$, and $\epsilon_t = t^{-2a}$, with $3/4 < a < 1$ and $1/2 < b < 2a - 1$. This holds, for instance, if $\alpha_t = t^{-5/6}$, $\beta_t = t_{-2/3}$, and $\epsilon_t = t^{-25/36}$. A key feature of this result is that the complexity of the function parameterization will not grow untenably large due to the use of our KOMP-based compression method which ties the sparsification bias ϵ_t to the algorithm step-size α_t. In particular, by modifying the above exact convergence result for diminishing step-sizes to one in which they are kept constant, we are able to keep constant compression budgets as well, and establish convergence to a neighborhood as well as the finiteness of the model order of Q, as stated next.

Theorem 2 *Consider the sequence z_t and $\{Q_t\}$ as stated in Algorithm 6. Assume the regularizer is positive $\lambda > 0$, Assumptions 1-3 hold, and the step-sizes are chosen as constant such that $0 < \alpha < \beta < 1$, with $\epsilon = C\alpha^2$ and the parsimony constant $C > 0$ is positive. Then the Bellman error converges to a neighborhood in expectation, i.e.:*

$$\liminf_{t\to\infty} \mathbb{E}[J(Q_t)] \leq \mathcal{O}\left(\frac{\alpha\beta}{\beta-\alpha}\left[1+\sqrt{1+\frac{\beta-\alpha}{\alpha\beta}\left(\frac{1}{\beta}+\frac{\beta^2}{\alpha^2}\right)}\right]\right) \tag{6.59}$$

Proof: Begin with the expression in (6.42), and substitute in constant step-size selections with $\epsilon = C\alpha^2$ and $(1-\beta) \leq 1$, and compute the total expectation ($\mathcal{F}_t = \mathcal{F}_0$) to write

$$\begin{aligned}\mathbb{E}[J(Q_{t+1})] \leq \mathbb{E}[J(Q_t)] - \alpha\left(1-\frac{\alpha G_Q^2}{\beta}\right)\mathbb{E}[\|\nabla_Q J(Q)\|^2] \\ + C\alpha^2\mathbb{E}[\|\nabla_Q J(Q_t)\|_{\mathcal{H}}] + \frac{\beta}{2}\mathbb{E}[(z_{t+1}-\bar{\delta}_t)^2] + \frac{L_Q\sigma_Q^2\alpha^2}{2}\,,\end{aligned} \tag{6.60}$$

From here, note that sequence $\mathbb{E}[(z_{t+1}-\bar{\delta}_t)^2]$ is identical (except re-written in terms of Q-functions rather than value functions) to the sequence in Lemma 1(iii) of [117], and therefore, analogous reasoning to that

which yields eqn. (86) in Appendix D of [117] allows us to write

$$\mathbb{E}[(z_{t+1} - \bar{\delta}_t)^2] \leq \frac{2L_Q}{\beta^2}\left[\alpha^2(G_\delta^2 G_Q^2 + \lambda^2 D^2\right] + 2\beta\sigma_\delta^2\,, \tag{6.61}$$

which follows from applying (6.40) to the Hilbert-norm difference of Q-functions term and recursively substituting the total expectation of (6.41) back into itself, and simplifying the resulting geometric sum. Now, we may substitute the right-hand side of (6.61) into the third term on the right-hand side of (6.60) to obtain

$$\begin{aligned}\mathbb{E}[J(Q_{t+1})] \leq \mathbb{E}[J(Q_t)] - \alpha\left(1 - \frac{\alpha G_Q^2}{\beta}\right)\mathbb{E}[\|\nabla_Q J(Q)\|^2] + C\alpha^2\mathbb{E}[\|\nabla_Q J(Q_t)\|_{\mathcal{H}}] \\ + \underbrace{\frac{2L_Q}{\beta}\left[\alpha^2(G_\delta^2 G_Q^2 + \lambda^2 D^2\right] + 2\beta^2\sigma_\delta^2 + \frac{L_Q\sigma_Q^2\alpha^2}{2}}_{:=K}\,,\end{aligned} \tag{6.62}$$

In the preceding expression we have defined the constant factor K that depends on algorithm step-sizes α and β. The rest of the proof proceeds as follows: we break the right-hand side of (6.62) into two subsequences, one in which the expected norm of the cost functional's gradient $\mathbb{E}[\|\nabla_Q J(Q_t)\|_{\mathcal{H}}]$ is below a specified threshold, whereby $J(Q_t)$ is a decreasing sequence in expectation, and one where this condition is violated. We can use this threshold condition to define a deterministically decreasing auxiliary sequence to which the Monotone Convergence Theorem applies, and hence we obtain convergence of the auxiliary sequence. Consequently, we obtain convergence in infimum of the expected value of $J(Q_t)$ to a neighborhood.

We proceed by defining the conditions under which $\mathbb{E}[J(Q_t)]$ is decreasing, i.e.,

$$\begin{aligned}\mathbb{E}[J(Q_{t+1})] &\leq \mathbb{E}[J(Q_t)] - \alpha\left(1 - \frac{\alpha G_Q^2}{\beta}\right)\mathbb{E}[\|\nabla_Q J(Q)\|^2] + C\alpha^2\mathbb{E}[\|\nabla_Q J(Q_t)\|_{\mathcal{H}}] + K \\ &\leq \mathbb{E}[J(Q_t)]\end{aligned} \tag{6.63}$$

Note that (6.63) holds whenever the following is true:

$$-\alpha\left(1 - \frac{\alpha G_Q^2}{\beta}\right)\mathbb{E}[\|\nabla_Q J(Q)\|^2] + C\alpha^2\mathbb{E}[\|\nabla_Q J(Q_t)\|_{\mathcal{H}}] + K \leq 0 \tag{6.64}$$

Observe that the left-hand side of (6.64) defines a quadratic function of $\mathbb{E}[\|\nabla_Q J(Q_t)\|_{\mathcal{H}}]$ which opens

downward. We can solve for the condition under which (6.64) holds with equality by obtaining the positive root (since $\mathbb{E}[\|\nabla_Q J(Q_t)\|_{\mathcal{H}}] \geq 0$) of this expression through the quadratic formula:

$$\begin{aligned}\mathbb{E}[\|\nabla_Q J(Q)\|] &= \left(C + \left[C^2 + 4\left(\frac{1}{\alpha} - \frac{G_Q^2}{\beta}\right)\left[\frac{2L_Q}{\beta}\left[(G_\delta^2 G_Q^2 + \lambda^2 D^2)\right] + \frac{2\beta^2\sigma_\delta^2}{\alpha^2} + \frac{L_Q\sigma_Q^2}{2}\right]\right]^{1/2}\right)\left(\frac{1}{\alpha} - \frac{G_Q^2}{\beta}\right)^{-1} \\ &= \mathcal{O}\left(\frac{\alpha\beta}{\beta-\alpha}\left[1 + \sqrt{1 + \frac{\beta-\alpha}{\alpha\beta}\left(\frac{1}{\beta} + \frac{\beta^2}{\alpha^2}\right)}\right]\right)\end{aligned}$$

where we have cancelled out a factor of α^2 as well as common factors of -1. Now, define the right-hand side of the previous expression as a constant Δ, and the auxiliary sequence

$$\Gamma_t = \mathbb{E}[J(Q_t)]\,\mathbb{1}\left\{\min_{u\leq t} -\alpha\left(1 - \frac{\alpha G_Q^2}{\beta}\right)\mathbb{E}[\|\nabla_Q J(Q)\|^2] + C\alpha^2\mathbb{E}[\|\nabla_Q J(Q_t)\|_{\mathcal{H}}] + K > \Delta\right\}$$

where $\mathbb{1}\{E\}$ denotes the indicator function of a (deterministic) event E. From here, we note that Γ_t is nonnegative since $J(Q_t) \geq 0$. Moreover, Γ_t is decreasing: either the indicator is positive, in which case its argument is true, and hence (6.64) holds with equality. When (6.64) holds with equality, the objective is decreasing, namely, (6.63) is valid. Alternatively, condition inside the indicator is null, which, due to the use of the minimum in the definition (D), means that the indicator is null for all subsequent times. Therefore, in either case, Γ_t is nonnegative and decreasing, and therefore we may apply the Monotone Convergence Theorem [132] to conclude $\Gamma_t \to 0$. Therefore, we have either that $\lim_t \mathbb{E}[J(Q_t)] - \Delta = 0$ or that the indicator in (D) is null for $t \to \infty$. Taken together, these statements allow us to conclude

$$\liminf_{t\to\infty}\ \mathbb{E}[J(Q_t)] \leq \mathcal{O}\left(\frac{\alpha\beta}{\beta-\alpha}\left[1 + \sqrt{1 + \frac{\beta-\alpha}{\alpha\beta}\left(\frac{1}{\beta} + \frac{\beta^2}{\alpha^2}\right)}\right]\right) \tag{6.65}$$

which is as stated in (6.59). ■

The expression on the right-hand side of (6.59) is a complicated posynomial of α and β, but is positive provided $\beta > \alpha$, and for a fixed β increases as α increases. This means that more aggressive selections of α, for a given β, yield a larger limiting lower bound on the Bellman error. A simple example which satisfies the constant step-size conditions $0 < \alpha < \beta < 1$ is $\beta = \alpha + \iota$ for some small constant $\iota > 0$. This is consistent with the diminishing step-size conditions where $\alpha_t/\beta_t \to 0$ means that α_t must be smaller than β_t which is

in $(0, 1)$.

In subsequent sections, we investigate the empirical validity of the proposed approach on two autonomous control tasks: the Inverted Pendulum and Continuous Mountain Car, for which observe consistent convergence in practice. To do so, first some implementation details of Algorithm 6 must be addressed.

E Experiments

We shift to experimentation of the previously developed methods. We benchmark the proposed algorithms on two classic control problems, the Continuous Mountain Car and the Inverted Pendulum, implemented by OpenAI Gym [145].

For Continuous Mountain Car problem, the state space is $p = 2$ dimensional, consisting of position and velocity, bounded within $[-1.2, 0.6]$ and $[-0.07, 0.07]$, respectively. The action space is $q = 1$ dimensional: force on the car, within the interval $[-1, 1]$. The reward function is 100 when the car reaches the goal at position 0.45, and $-0.1a^2$ for any action a. For each episode, the start position of the car is initialized uniformly at random in $[-0.6, 0.4]$.

In the Inverted Pendulum problem, the state space is $p = 3$ dimensional, consisting of the sine of the angle of the pendulum, the cosine of the angle, and the angular velocity, bounded within $[-1.0, 1.0]$, $[-1.0, 1.0]$, and $[-8.0, 8.0]$ respectively. The action space is $q = 1$ dimensional: joint effort, within the interval $[-2.0, 2.0]$. The reward function is $r(\theta, \dot{\theta}, a) = -(\theta^2 + 0.1\dot{\theta}^2 + 0.001a^2)$, where θ is the angle of the pendulum relative to vertical, and $\dot{\theta}$ is the angular velocity. The goal of the problem is to balance the pendulum at the unstable equilibrium where $\theta = 0$.

For all experiments with the Continuous Mountain Car and the Inverted Pendulum problems, we used Gaussian kernels with a fixed non-isotropic bandwidth. The relevant parameters are the step-sizes α and β, the regularizer λ, and the approximation error constant, C, where we fix the compression budget $\epsilon = C\alpha^2$. These learning parameters were tuned through a grid search procedure, and are summarized in Table 6.1.

We investigate two methods for exploration as the agent traverses the environment. When using an exploratory policy, actions are selected uniformly at random from the action space. When using an ρ-greedy policy, we select actions randomly with probability 1, which linearly decays to 0.1 after 10^5 exploratory training steps. In addition, we explore the use of a replay buffer. This method re-reveals past data to the agent

uniformly at random. For the Mountain Car problem, we also use prioritized memory which replays samples based on the magnitude of their temporal action difference.

A comprehensive summary of our experimental results may be found in Table 6.1. We bold which methods perform best across many different experimental settings. Interestingly, playback buffers play a role in improving policy learning in the Pendulum domain but not for Mountain Car, due to the difference in their reward structure.

For completeness of comparison, we also consider two alternative ways to construct the kernel parameterization: one based on a compositional extension of the fixed-subspace stochastic gradient scheme in [128], and another based on a direct discretization of the space into bins of a fixed size inversely proportional to the number of bins raised to the power of the dimension of the space. Respectively denote these methods as "Budgeted" and "Discretized." For these comparators, we hand-tuned parameters to obtain the best performance, which unsurprisingly were similar to those which worked well for KQ Learning in its various forms. We further attempted to include a fixed linear basis expansion akin to [146] but were unable to obtain reasonable performance. Overall, KQ Learning performs comparably to the "Budgeted" approach on Pendulum with a smaller standard deviation, and better than the "Discretized" approach with 1000 bins on both environments by a substantial margin.

In Figure 6.1, we spotlight the results of this experiment in the Continuous Mountain Car domain for the Hybrid algorithm that is described in Appendix B. We plot the normalized Bellman test error Fig. 6.1b, defined by the sample average approximation of (6.6) divided by the Hilbert norm of Q_t over a collection of test trajectories, as well as the average rewards during training (Fig. 6.1a), and the model order, i.e., the number of training examples in the kernel dictionary (Fig. 6.1c), all relative to the number of training samples processed. Observe that the Bellman test error converges and the interval average rewards approach 90, which is comparable to top entries on the OpenAI Leaderboard [145], such as Deep Deterministic Policy Gradient [147] . Moreover, we obtain this result with a complexity reduction orders of magnitude relative to existing methods for Q-function and policy representation. This trend is corroborated for the Inverted Pendulum in Figure 6.2: the normalized Bellman error converges and the model complexity remains moderate. Also, the average rewards approach -200.

A feature of our method is the interpretability of the resulting Q function, which we use to plot the value function (6.3a) and policy (6.3b). One key metric is the coverage of the kernel points in the state-action space.

Count	Environment	Algorithm	Replay Buffer	Policy	Steps	α	β	C	Kernel Σ	Order	Training Loss	Test Rewards
1	Inv. Pendulum	KQ	Yes	Exploratory	100K	0.25	1.00	2.00	[0.5,0.5,2,0.5]	29.53	1.28±5.66	-1427.30±188.86
2	Inv. Pendulum	KQ	Yes	ρ-greedy	100K	0.25	1.00	2.00	[0.5,0.5,2,0.5]	40.43	22.58±17.88	-1352.96±220.65
3	Inv. Pendulum	Hybrid	Yes	ρ-greedy	500K	0.25	1.00	2.00	[0.5,0.5,2,0.5]	193.01	**7.56 ±25.74**	**-169.78 ±79.02**
4	Inv. Pendulum	Budgeted	Yes	ρ-greedy	500K	0.25	1.00	2.00	[0.5,0.5,2,0.5]	200	**6.91 ±24.63**	**-121.74 ± 81.97**
5	Inv. Pendulum	Discretized	Yes	ρ-greedy	500K	0.25	1.00	2.00	[0.5,0.5,2,0.5]	2048	25.47 ±26.96	-1274.41 ± 218.93
6	Inv. Pendulum	SG	Yes	ρ-greedy	200K	0.25	1.00	2.00	[0.5,0.5,2,0.5]	201.75	**4.17 ± 20.97**	**-150.36**
7	Inv. Pendulum	KQ	No	Exploratory	100K	0.25	1.00	2.00	[0.5,0.5,2,0.5]	40.74	0.86±2.74	-1259.63±384.24
8	Inv. Pendulum	KQ	No	ρ-greedy	100K	0.25	1.00	2.00	[0.5,0.5,2,0.5]	79.72	19.11±18.49	-1497.66±84.72
9	Inv. Pendulum	Hybrid	No	ρ-greedy	500K	0.25	1.00	2.00	[0.5,0.5,2,0.5]	234.00	1.82 ±10.64	-115.01±78.57
10	Inv. Pendulum	SG	No	ρ-greedy	200K	0.25	1.00	2.00	[0.5,0.5,2,0.5]	241.08	106.41 ±748.56	-403.89±76.98
11	Cont. M. Car	KQ	Prioritized	Exploratory	100K	0.25	1.00	0.10	[0.8,0.07,1.0]	86.07	15.17±85.19	-10.58±1.57
12	Cont. M. Car	KQ	Prioritized	ρ-greedy	500K	0.25	1.00	0.10	[0.8,0.07,1.0]	97.0	**2.04±7.54**	**93.09±3.03**
13	Cont. M. Car	Budgeted	Prioritized	ρ-greedy	500K	0.25	1.00	0.10	[0.8,0.07,1.0]	100	**3.28 ± 21.43**	**78.37 ± 35.71**
14	Cont. M. Car	Discretized	Prioritized	ρ-greedy	500K	0.25	1.00	0.10	[0.8,0.07,1.0]	1000	**200.72 ±427.13**	**75.86±40.40**
15	Cont. M. Car	Hybrid	Prioritized	ρ-greedy	500K	0.25	1.00	0.10	[0.8,0.07,1.0]	80.0	**1.33±4.33**	**93.71±2.53**
16	Cont. M. Car	SG	Prioritized	ρ-greedy	500K	0.25	1.00	0.10	[0.8,0.07,1.0]	111.41	77.30±582.26	61.57±49.95
17	Cont. M. Car	KQ	No	Exploratory	100K	0.25	1.00	0.10	[0.8,0.07,1.0]	57.55	0.13±3.58	-2.26±2.01
18	Cont. M. Car	KQ	No	ρ-greedy	500K	0.25	1.00	0.10	[0.8,0.07,1.0]	78.73	0.83±19.88	-29.70±0.58
19	Cont. M. Car	Hybrid	No	ρ-greedy	500K	0.25	1.00	0.10	[0.8,0.07,1.0]	84.97	**0.38 ±3.18**	**90.18 ±26.74**
20	Cont. M. Car	SG	No	ρ-greedy	500K	0.25	1.00	0.10	[0.8,0.07,1.0]	81.04	**0.80±7.98**	**95.07±1.13**

Table 6.1: A summary of parameter selection details for our comparison of KQ-Learning(KQ), Hybrid, and Semi-Gradient(SG) methods. In the right-most column, we display the model order and training loss (Bellman error) over averaged over 1000 training steps with standard deviations. Moreover, we report the accumulation of rewards during testing averaged over 20 episodes, again with standard deviations per episode computed over this range. The best results for each problem setting are bolded for emphasis, which "solve" the problem according to reward benchmarks set by OpenAI. We observe the replay buffer improves learning in the Pendulum domain but yields little benefit in the Mountain Car problem. Overall, KQ Learning performs comparably to the "Budgeted" approach on Pendulum with a lower standard deviation, and better than the "Discretized" approach with 1000 bins on both environments by a substantial margin.

We can make conclusions about the importance of certain parts of the space for obtaining as much value as possible by the density of the model points throughout the space, which may have importance in mechanical or econometric applications.

F Proof of Proposition 1

Proof 3 *Consider the RKHS norm difference of the projected and un-proejcted stochastic quasi-gradient defined by* (6.29) *and* (6.30)

$$\begin{aligned}
&\|\tilde{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_t') - \hat{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_t')\|_{\mathcal{H}}^2 \\
&\qquad = \|(Q_t - \mathcal{P}_{\mathcal{H}_{\mathbf{D}_{t+1}}}[Q_t - \alpha_t \hat{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_t')])/\alpha_t - \hat{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_t')\|_{\mathcal{H}}^2
\end{aligned}$$

Multiply and divide $\hat{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)$ *by* α_t *and reorder terms to write*

$$\begin{aligned}
&\| \frac{(Q_t - \alpha_t \hat{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t))}{\alpha_t} - \frac{(\mathcal{P}_{\mathcal{H}_{\mathbf{D}_{t+1}}}[Q_t - \alpha_t \hat{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)])}{\alpha_t} \|^2_{\mathcal{H}} \\
&\quad = \frac{1}{\alpha_t^2} \| (Q_t - \alpha_t \hat{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)) - (\mathcal{P}_{\mathcal{H}_{\mathbf{D}_{t+1}}}[Q_t - \alpha_t \hat{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)]) \|^2_{\mathcal{H}} \\
&\quad\quad = \frac{1}{\alpha_t^2} \| \tilde{Q}_{t+1} - Q_{t+1} \|^2_{\mathcal{H}} \leq \frac{\epsilon_t^2}{\alpha_t^2}
\end{aligned}$$

where we have pulled the nonnegative scalar α_t *outside of the norm on the second line and substituted the definition of* $\tilde{Q}_{t+1}$ *and* Q_{t+1}*. We also apply the KOMP residual stopping criterion from Algorithm 2,* $\| \tilde{Q}_{t+1} - Q_{t+1} \| \leq \epsilon_t$ *to yield* (6.39)*.*

G Proof of Lemma 1

Proof 4 *Lemma 1*(1) *Consider the Hilbert-norm difference of action-value functions at the next and current iteration and use the definition of* Q_{t+1}

$$\| Q_{t+1} - Q_t \|^2_{\mathcal{H}} = \alpha_t^2 \| \tilde{\nabla}_Q J(Q_t, z_{t+1}; (\mathbf{s}_t, \mathbf{a}_t), (\mathbf{s}'_t, \mathbf{a}'_t) \|^2_{\mathcal{H}} \tag{6.66}$$

We add and subtract the functional stochastic quasi-gradient $\hat{\nabla}_Q J(Q_t, z_{t+1}; (\mathbf{s}_t, \mathbf{a}_t), (\mathbf{s}'_t, \mathbf{a}'_t)$ *from* (6.66) *and apply the triangle inequality* $(a+b)^2 \leq 2a^2 + 2b^2$ *which holds for any* $a, b > 0$*.*

$$\begin{aligned}
\| Q_{t+1} - Q_t \|^2_{\mathcal{H}} \leq\ & 2\alpha_t^2 \| \hat{\nabla}_Q J(Q_t, z_{t+1}; (\mathbf{s}_t, \mathbf{a}_t), (\mathbf{s}'_t, \mathbf{a}'_t) \|^2_{\mathcal{H}} \\
& + 2\alpha_t^2 \| \hat{\nabla}_Q J(Q_t, z_{t+1}; (\mathbf{s}_t, \mathbf{a}_t), (\mathbf{s}'_t, \mathbf{a}'_t) - \tilde{\nabla}_Q J(Q_t, z_{t+1}; (\mathbf{s}_t, \mathbf{a}_t), (\mathbf{s}'_t, \mathbf{a}'_t) \|^2_{\mathcal{H}}
\end{aligned} \tag{6.67}$$

Now, we may apply Proposition 1 to the second term. Doing so and computing the expectation conditional on the filtration $\mathcal{F}_t$ *yields*

$$\mathbb{E}[\| Q_{t+1} - Q_t \|^2_{\mathcal{H}}] \mid \mathcal{F}] = 2\alpha_t^2 \mathbb{E}[\| \hat{\nabla}_Q J(Q_t, z_{t+1}; (\mathbf{s}_t, \mathbf{a}_t), (\mathbf{s}'_t, \mathbf{a}'_t) \|^2_{\mathcal{H}} \mid \mathcal{F}_t] + 2\epsilon_t^2$$

Using the Cauchy-Schwarz inequality together with the Law of Total Expectation and the definition of the functional stochastic quasi-gradient to upper estimate the first term on the right-hand side of (4) *as*

$$\begin{aligned}&\mathbb{E}[\|Q_{t+1} - Q_t\|_{\mathcal{H}}^2 \mid \mathcal{F}_t] \\ &\quad \leq 2\alpha_t^2 \mathbb{E}\{\|\gamma\kappa((\mathbf{s}_t', \mathbf{a}_t'), \cdot) - \kappa((\mathbf{s}_t, \mathbf{a}_t), \cdot)\|_{\mathcal{H}}^2 \mathbb{E}[z_{t+1}^2 \mid \mathbf{s}_t, a_t] \mid \mathcal{F}_t\} + 2\alpha_t^2 \lambda \|Q_t\|_{\mathcal{H}}^2 + 2\epsilon_t^2 \end{aligned} \tag{6.68}$$

Now, use the fact that z_{t+1} *has a finite second conditional moment [cf.* (6.33)*], yielding*

$$\mathbb{E}[\|Q_{t+1} - Q_t\|_{\mathcal{H}}^2 \mid \mathcal{F}_t] \leq 2\alpha_t^2 G_\delta^2 \mathbb{E}[\|\gamma\kappa((\mathbf{s}_t', \mathbf{a}_t'), \cdot) - \kappa((\mathbf{s}_t, \mathbf{a}_t), \cdot)\|_{\mathcal{H}}^2 \mid \mathcal{F}_t] + 2\alpha_t^2 \lambda \|Q_t\|_{\mathcal{H}}^2 + 2\epsilon_t^2$$

From here, we may use the fact that the functional gradient of the temporal action-difference $\gamma\kappa((\mathbf{s}_t', \mathbf{a}_t'), \cdot) - \kappa((\mathbf{s}_t, \mathbf{a}_t), \cdot)$ *has a finite second conditional moment* (6.35) *and that the Q function sequence is bounded* (6.38) *to write:*

$$\mathbb{E}[\|Q_{t+1} - Q_t\|_{\mathcal{H}}^2 \mid \mathcal{F}_t] \leq 2\alpha_t^2 (G_\delta^2 G_Q^2 + \lambda^2 D^2) + 2\epsilon_t^2 \tag{6.69}$$

which is as stated in Lemma 1(1)*.*

Proof 5 *Lemma 1*(2) *Begin by defining the scalar quantity* e_t *as the difference of mean temporal-action differences scaled by the forgetting factor* β_t*, i.e.* $e_t = (1 - \beta_t)(\bar{\delta}_t - \bar{\delta}_{t-1})$*. Then, we analyze the auxiliary variable* z_{t+1} *with respect to the conditional mean temporal action difference* $\bar{\delta}_t$*, plus the difference of the mean temporal differences:*

$$z_{t+1} - \bar{\delta}_t + e_t = (1 - \beta_t) z_t + \beta_t \delta_t - [(1 - \beta_t)\bar{\delta}_t + \beta_t \bar{\delta}_t] + (1 - \beta_t)(\bar{\delta}_t - \bar{\delta}_{t-1}) \tag{6.70}$$

where we make use of the definition of z_{t+1}*, the fact that* $\bar{\delta}_t = \{(1 - \beta_t)\bar{\delta}_t + \beta_t \bar{\delta}_t\}$ *and the definition of* e_t *on the right-hand side of* (6.70)*. Observe that the result then simplifies to* $z_{t+1} - \bar{\delta}_t + e_t = (1 - \beta_t) z_t + \beta_t (\bar{\delta}_t - \bar{\delta}_{t-1})$ *by grouping like terms and canceling the redundant* $\bar{\delta}_t$*. Squaring* (6.70)*, using this simplification, yields*

$$(z_{t+1} - \bar{\delta}_t + e_t)^2 = (1 - \beta_t)^2 (z_t - \bar{\delta}_{t-1})^2 + \beta_t^2 (\delta_t - \bar{\delta}_t)^2 + 2(1 - \beta_t)\beta_t (z_t - \bar{\delta}_{t-1})(\delta_t - \bar{\delta}_t)$$

Now, we compute the expectation conditioned on the algorithm history $\mathcal{F}_t$ to write

$$\begin{aligned}\mathbb{E}[(z_{t+1} - \bar{\delta}_t + e_t)^2 \mid \mathcal{F}_t] =& (1-\beta_t)^2(z_t - \bar{\delta}_{t-1})^2 + \beta_t^2 \mathbb{E}[(\delta_t - \bar{\delta}_t)^2 \mid \mathcal{F}_t] \\ &+ 2(1-\beta_t)\beta_t(z_t - \bar{\delta}_{t-1})\mathbb{E}[(\delta_t - \bar{\delta}_t) \mid \mathcal{F}_t] \end{aligned} \tag{6.71}$$

We apply the assumption that the temporal action difference δ_t is an unbiased estimator for its conditional mean $\bar{\delta}_t$ with finite variance (Assumption 2) to write

$$\mathbb{E}[(z_{t+1} - \bar{\delta}_t + e_t) \,\big|\, \mathcal{F}_t] = (1-\beta_t)^2(z_t - \bar{\delta}_{t-1})^2 + \beta_t^2 \sigma_\delta^2 \tag{6.72}$$

We obtain an upper estimate on the conditional mean square of $z_{t+1} - \bar{\delta}_t$ by using the inequality $\|a+b\|^2 \leq (1+\rho)\|a\|^2 + (1+1\rho)\|b\|^2$ which holds for any $\rho > 0$: set $a = z_{t+1} - \bar{\delta}_t + e_t$, $b = -e_t$, $\rho = \beta_t$ to write

$$(z_{t+1} - \bar{\delta}_t)^2 \leq (1+\beta_t)(z_{t+1} - \bar{\delta}_t + e_t)^2 + \left(1 + \frac{1}{\beta_t}\right) e_t^2 \tag{6.73}$$

Observe that (6.73) *provides an upper-estimate of the square sub-optimality $(z_{t+1} - \bar{\delta}_t)^2$ in terms of the squared error sequence $(z_{t+1} - \bar{\delta}_t + e_t)^2$. Therefore, we compute the expectation of* (6.73) *and substitute* (6.72) *for the terms involving the error sequence $(z_{t+1} - \bar{\delta}_t + e_t)^2$, which results in gaining a factor of $(1+\beta_t)$ on the right-hand side. Collecting terms yields*

$$\mathbb{E}[(z_{t+1} - \bar{\delta}_t)^2 \mid \mathcal{F}_t] = (1+\beta_t)[(1-\beta_t)^2(z_t - \bar{\delta}_{t-1})^2 + \beta_t^2\sigma_\delta^2] + \left(\frac{1+\beta_t}{\beta_t}\right) e_t^2$$

Using $(1-\beta_t^2)(1-\beta_t) \leq (1-\beta_t)$ for the first term and $(1-\beta_t)\beta_t^2 \leq 2\beta_t^2$ for the second:

$$\mathbb{E}[(z_{t+1} - \bar{\delta}_t)^2 \mid \mathcal{F}_t] = (1-\beta_t)(z_t - \bar{\delta}_{t-1})^2 + 2\beta_t^2\sigma_\delta^2 + \left(\frac{1+\beta_t}{\beta_t}\right) e_t^2 \tag{6.74}$$

We can bound the term involving e_t, which represents the difference of mean temporal differences. By definition, we have $|e_t| = (1-\beta_t)|(\bar{\delta}_t - \bar{\delta}_{t-1})|$:

$$(1-\beta_t)|(\bar{\delta}_t - \bar{\delta}_{t-1})| \leq (1-\beta_t)L_Q\|Q_t - Q_{t-1}\|_{\mathcal{H}}\,, \tag{6.75}$$

where we apply the Lipschitz continuity of the temporal difference with respect to the action-value function [cf. (6.37)*]. Substitute the right-hand side of* (6.75) *and simplify the expression in the last term as* $(1-\beta_t^2)/\beta_t \leq 1/\beta_t$ *to conclude* (6.42)*.*

Proof 6 *Lemma 1*(3) *Following the proof of Theorem 4 of [125], consider the Taylor expansion of* $J(Q)$ *and applying the fact that it has Lipschitz continuous functional gradients to upper-bound the second-order terms. Doing so yields the quadratic upper bound:*

$$J(Q_{t+1}) \leq J(Q_t) + \langle \nabla J(Q_t), Q_{t+1} - Q_t \rangle_{\mathcal{H}} + \frac{L_Q}{2} \|Q_{t+1} - Q_t\|_{\mathcal{H}}^2 \,. \tag{6.76}$$

Substitute the fact that the difference between consecutive action-value functions is the projected quasi-stochastic gradient $Q_{t+1} - Q_t = -\alpha_t \tilde{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_t')$ (6.31) *into* (6.76) *.*

$$J(Q_{t+1}) \leq J(Q_t) - \alpha_t \langle \nabla J(Q_t), \tilde{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_t') \rangle_{\mathcal{H}} + \frac{L_Q \alpha_t^2}{2} \|\tilde{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_t')\|_{\mathcal{H}}^2 \,.$$

Subsequently, we use the short-hand notation

$$\hat{\nabla}_Q J(Q_t) := \hat{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_t')$$
$$\tilde{\nabla}_Q J(Q_t) := \tilde{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_t')$$

for the stochastic and projected stochastic quasi-gradients, (6.29) *and* (6.30)*, respectively. Now add and subtract the inner-product of the functional gradient of* J *with the stochastic gradient, scaled by the step-size* $\alpha_t \langle \nabla J(Q_t), \hat{\nabla}_Q J(Q_t, z_{t+1}; \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_t') \rangle_{\mathcal{H}}$*, and* $\alpha_t \|\nabla_Q J(Q_t)\|^2$ *into above expression and gather terms.*

$$\begin{aligned} J(Q_{t+1}) \leq J(Q_t) &- \alpha_t \|\nabla_Q J(Q)\|^2 + \frac{L_Q \alpha_t^2}{2} \|\tilde{\nabla}_Q J(Q_t)\|_{\mathcal{H}}^2 \\ &- \alpha_t \langle \nabla J(Q_t), \tilde{\nabla}_Q J(Q_t) - \hat{\nabla}_Q J(Q_t) \rangle_{\mathcal{H}} + \alpha_t \langle \nabla J(Q_t), \nabla J(Q_t) - \hat{\nabla}_Q J(Q_t) \rangle_{\mathcal{H}} \end{aligned} \tag{6.77}$$

The last two terms on the right-hand side of (6.77) *are terms associated with the directional error between the true gradient and the stochastic quasi-gradient, as well as the stochastic quasi-gradient with respect to the projected stochastic quasi-gradient. The former term may be addressed through the KOMP stopping criterion in Proposition 1, whereas the later may be analyzed through the Law of Total Expectation and Assumptions*

2 - 3.

We proceed to address the second term on the right-hand side of (6.77) *by applying Cauchy-Schwarz to write*

$$|-\alpha_t\langle\nabla J(Q_t),\tilde{\nabla}_Q J(Q_t)-\hat{\nabla}_Q J(Q_t)\rangle_{\mathcal{H}}|\leq\alpha_t\|\nabla J(Q_t)\|_{\mathcal{H}}\|\tilde{\nabla}_Q J(Q_t)-\hat{\nabla}_Q J(Q_t)\|_{\mathcal{H}} \tag{6.78}$$

Now, apply Proposition 1 to $\|\tilde{\nabla}_Q J(Q_t)-\hat{\nabla}_Q J(Q_t)\|_{\mathcal{H}}$*, the Hilbert-norm error induced by sparse projections on the right-hand side of* (6.78) *and cancel the factor of* α_t*:*

$$\alpha_t\langle\nabla J(Q_t),\tilde{\nabla}_Q J(Q_t)-\hat{\nabla}_Q J(Q_t)\rangle_{\mathcal{H}}\leq\epsilon_t\|\nabla J(Q_t)\|_{\mathcal{H}} \tag{6.79}$$

Next, we address the last term on the right-hand side of (6.77)*. To do so, we will exploit Assumptions 2 - 3 and the Law of Total Expectation. First, consider the expectation of this term, ignoring the multiplicative step-size factor, while applying* (6.34)*:*

$$\begin{aligned}\mathbb{E}\big[\nabla J(Q_t),\nabla_Q J(Q_t)-\hat{\nabla}_Q J(Q_t)\rangle_{\mathcal{H}}\,\big|\,\mathcal{F}_t\big]&\\ =\Big\langle\nabla_Q J(Q_t),\mathbb{E}[(\gamma\kappa((\mathbf{s}_t',\mathbf{a}_t'),\cdot)-\kappa((\mathbf{s}_t,\mathbf{a}_t),\cdot))(\bar{\delta}_t-z_{t+1})\,\big|\,\mathcal{F}_t]\Big\rangle_{\mathcal{H}}&\end{aligned} \tag{6.80}$$

In (6.80)*, we pull the expectation inside the inner-product, using the fact that* $\nabla_Q J(Q)$ *is deterministic. Note on the right-hand side of* (6.80)*, by using* (6.34)*, we have* $\bar{\delta}_t$ *inside the expectation in the above expression rather than a realization* δ_t*. Now, apply Cauchy-Schwartz to the above expression to obtain*

$$\begin{aligned}&\Big\langle\nabla_Q J(Q_t),\mathbb{E}[(\gamma\kappa((\mathbf{s}_t',\mathbf{a}_t'),\cdot)-\kappa((\mathbf{s}_t,\mathbf{a}_t),\cdot))(\bar{\delta}_t-z_{t+1})\,\big|\,\mathcal{F}_t]\Big\rangle_{\mathcal{H}}\\ &\quad\leq\|\nabla_Q J(Q_t)\|_{\mathcal{H}}\mathbb{E}[\|(\gamma\kappa((\mathbf{s}_t',\mathbf{a}_t'),\cdot)-\kappa((\mathbf{s}_t,\mathbf{a}_t),\cdot))\|_{\mathcal{H}}\times|\bar{\delta}_t-z_{t+1}|\,\big|\,\mathcal{F}_t]\end{aligned} \tag{6.81}$$

From here, apply the inequality $ab\leq\frac{\rho}{2}a^2+\frac{1}{2\rho}b^2$ *for* $\rho>0$ *with*

$$a=|\bar{\delta}_t-z_{t+1}|,\quad b=\alpha_t\|\nabla_Q J(Q_t)\|\|\gamma\kappa((\mathbf{s}_t',\mathbf{a}_t'),\cdot)-\kappa((\mathbf{s}_t,\mathbf{a}_t),\cdot)\|_{\mathcal{H}}, \tag{6.82}$$

and $\rho = \beta_t$ to the preceding expression:

$$\begin{aligned}
&\|\nabla_Q J(Q_t)\|_{\mathcal{H}} \mathbb{E}[\|(\gamma\kappa((\mathbf{s}_t', \mathbf{a}_t'), \cdot) - \kappa((\mathbf{s}_t, \mathbf{a}_t), \cdot))\|_{\mathcal{H}} |\bar{\delta}_t - z_{t+1}| \,\big|\, \mathcal{F}_t]\Big\rangle_{\mathcal{H}} \\
&\quad \leq \frac{\beta_t}{2}\mathbb{E}[(\bar{\delta}_t - z_{t+1})^2 \mid \mathcal{F}_t] + \frac{\alpha_t^2}{2\beta_t}\|\nabla J(Q_t)\|_{\mathcal{H}}^2 \mathbb{E}[\|\gamma\kappa((\mathbf{s}_t', \mathbf{a}_t'), \cdot) - \kappa((\mathbf{s}_t, \mathbf{a}_t), \cdot)\|_{\mathcal{H}}^2 \,\big|\, \mathcal{F}_t] \qquad (6.83)
\end{aligned}$$

To (6.83)*, we apply Assumption 3 regarding the finite second conditional of the difference of reproducing kernel maps* (6.35) *to the second term (together with the Law of Total Expectations and the fact that $\{\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_t'\} \subset \mathcal{F}_t$). When substituted into the right-hand side of the expectation of* (6.77) *conditional on $\mathcal{F}_t$, we obtain*

$$\begin{aligned}
\mathbb{E}[J(Q_{t+1}) \,\big|\, \mathcal{F}_t] \leq J(Q_t) - \alpha_t \left(1 - \frac{\alpha_t G_Q^2}{\beta_t}\right) \|\nabla_Q J(Q)\|^2 + \frac{\beta_t}{2}\mathbb{E}[(\bar{\delta}_t - z_{t+1})^2 \mid \mathcal{F}_t] \\
+ \frac{L_Q \sigma_Q^2 \alpha_t^2}{2} + \epsilon_t \|\nabla_Q J(Q_t)\|_{\mathcal{H}} \,, \qquad (6.84)
\end{aligned}$$

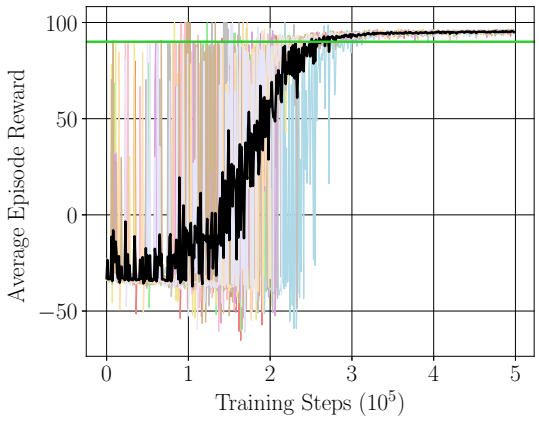
where we have also applied the fact that the projected stochastic quasi-gradient has finite conditional variance (6.36) *and gathered like terms to conclude* (6.42)*.*

Lemma 1 is may be seen as a nonparametric extension of Lemma 2 and A.1 of [125], or an extension of Lemma 6 in [117] to the non-convex case. Now, we may use Lemma 1 to connect the function sequence generated by Algorithm 6 to a special type of stochastic process called a coupled supermartingale, and therefore prove that Q_t converges to a stationary point of the Bellman error with probability 1. To our knowledge, this result is unique.

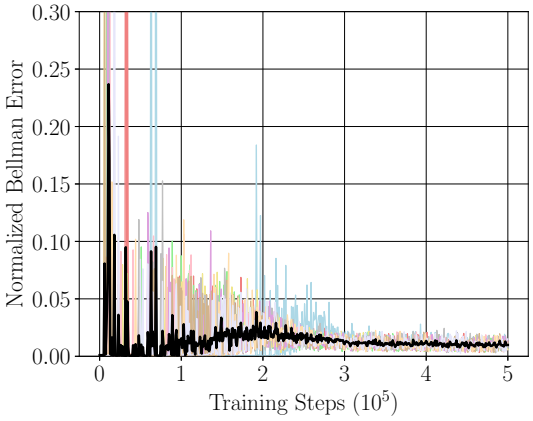
H Summary

We extended the nonparametric optimization approaches in [117] from policy evaluation to policy learning in continuous Markov Decision Problems. We reformulate policy learning defined by the Bellman optimality equation as a non-convex function-valued stochastic program with nested expectations. We hypothesize that the Bellman fixed point belongs to a reproducing Kernel Hilbert Space, motivated by the efficient semi-parametric form. By applying functional stochastic quasi-gradient method with greedily constructed subspace projections, we derived a new efficient variant of Q learning which is guaranteed to converge almost surely in continuous spaces, one of the first results of this type.

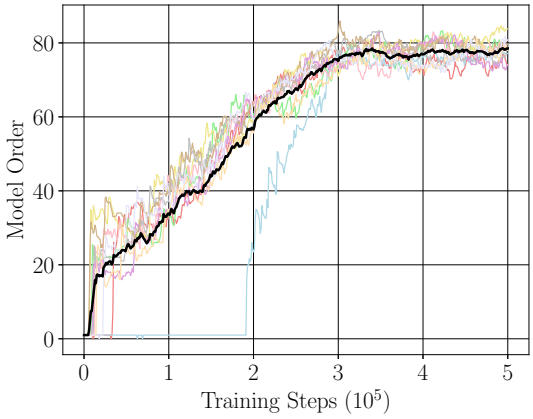
Unlike in the policy evaluation setting, in policy learning, we are forced to confront fundamental limitations associated with non-convexity and the explore-exploit tradeoff. We adopt a hybrid policy learning situation in which some actions are chosen greedily and some are chosen randomly. Through tuning of the proportion of actions that are greedy versus exploratory, we are able to design a variant of Q learning which learns good policies on some benchmark tasks, namely, the Continuous Mountain Car and the Inverted Pendulum, with orders of magnitude fewer training examples than existing approaches based on deep learning. Further, owing to the kernel parameterization of our learned Q functions, they are directly interpretable: the training points which are most vital for representing the minimal Bellman error action-value function are retained and define its feature representation.



(a) Average Training Reward

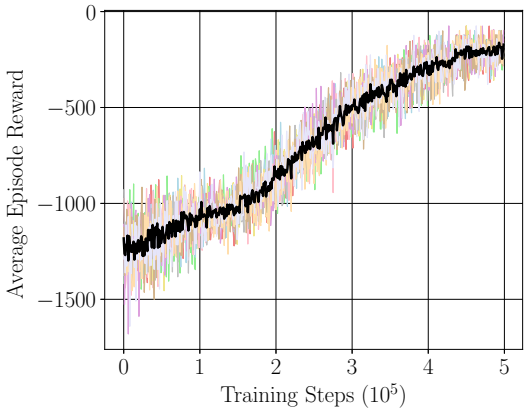


(b) Normalized Bellman Error

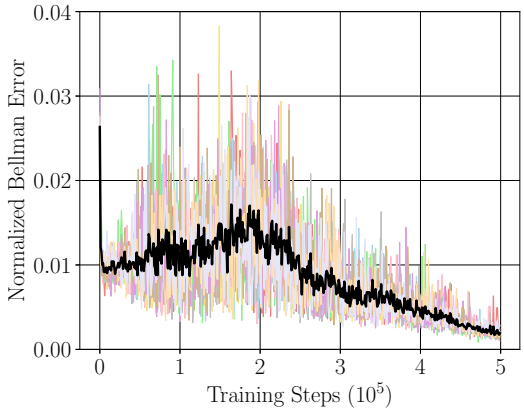


(c) Model Order of Q

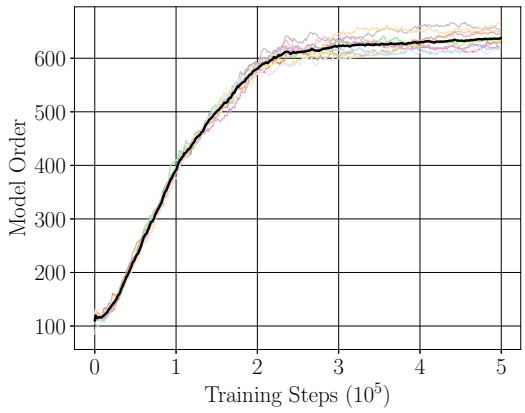
Figure 6.1. Results of 10 experiments over 500, 000 training steps were averaged (black curve) to demonstrate the learning progress for **Continuous Mountain Car** using the Hybrid algorithm with no replay buffer, Row 15 in Table 6.1. Fig. 6.1a shows the average reward obtained by the ρ-greedy policy during training. An average reward over 90 (green) indicates that we have solved Continuous Mountain Car, steering towards the goal location. Fig. 6.1b shows the normalized Bellman error during training, which converges to a small non-zero value. Fig. 6.1c shows the number of points parameterizing the kernel dictionary of Q during training, which remains under 80 on average. Overall, we solve Continuous Mountain Car with a complexity reduction by orders of magnitude relative to existing methods [2, 3].



(a) Average Training Reward

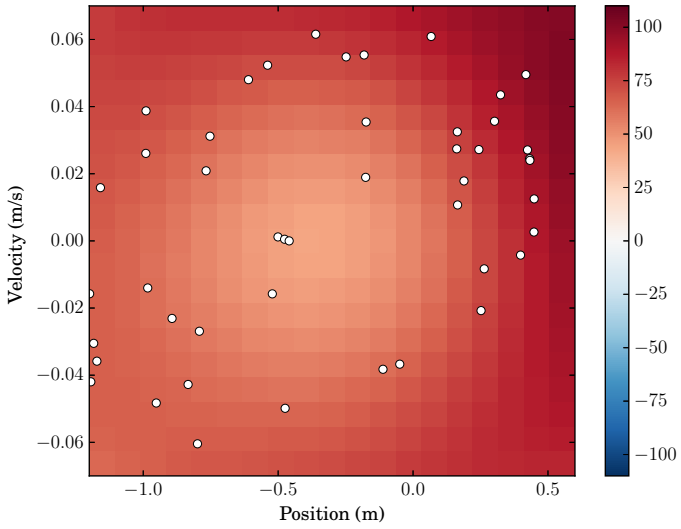


(b) Training Bellman Error

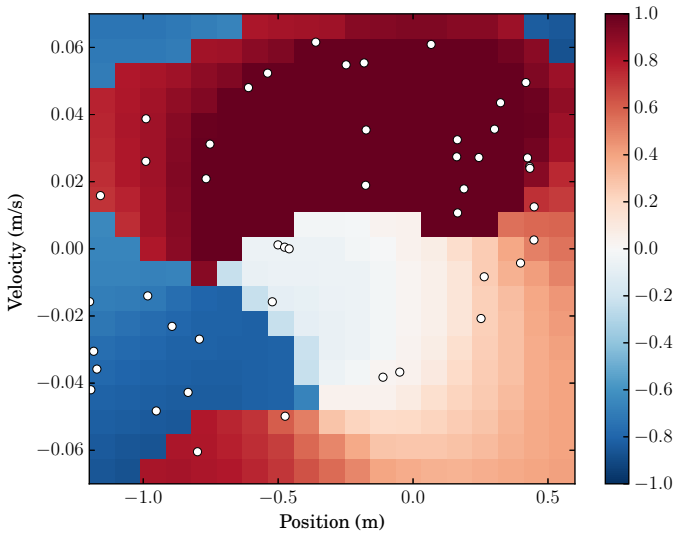


(c) Model Order of Q

Figure 6.2. Results of 10 experiments over 500, 000 training steps were averaged (black curve) to demonstrate the learning progress for the effective, convergent, and parsimonious solution for the **Pendulum** domain using the Hybrid algorithm with a replay buffer, Row 3 in Table 6.1. Fig. 6.2a shows the average reward obtained by the ρ-greedy policy during training. Fig. 6.2b shows the Bellman error for training samples (6.6) normalized by the Hilbert norm of Q, which converges to a small non-zero value. Fig. 6.2c shows the number of points parameterizing the kernel dictionary of Q during training, which remains under 700 on average. Overall, we solve Pendulum with a model complexity reduction by orders of magnitude relative to existing methods [2, 3], with a much smaller standard deviation around the average reward accumulation, meaning that these results are replicable.



(a) Value function $V(\mathbf{s})$ via limiting $Q(\mathbf{s}, \mathbf{a})$



(b) Policy $\pi(\mathbf{s})$ derived from limiting $Q(\mathbf{s}, \mathbf{a})$

Figure 6.3. For the **Continuous Mountain Car** problem, the learned Q-function is easily interpretable: we may visualize the value function, $V(s) = \max_a Q(s, a)$ (6.3a) and corresponding policy $\pi(s) = \text{argmax}_a Q(s, a)$ (6.3b). In Fig. 6.3a, the color indicates the value of the state, which is highest (dark red) near the goal 0.6. At this position, for any velocity, the agent receives an award of 100 and concludes the episode. In Fig. 6.3b, the color indicates the force on the car (action), for a given position and velocity (state). The learned policy takes advantage of the structure of the environment to accelerate the car without excess force inputs. The dictionary points are pictured in white and provide coverage of the state-action space.

Chapter 7

Composable Learning with Sparse Kernel Representations

A key goal in the design of distributed robot teams is the ability to learn collaboratively, so that knowledge and experience gained by one system can be seamlessly incorporated in another. Parallel experiences have already been used to stabilize and hasten joint learning [148], but we argue that there is also a need for learning techniques designed for loosely-coupled teams, in which members may be disconnected for long periods of time and then briefly reconnect to share information and coordinate. We re-consider joint learning problems where coordination is now an infrequent event and not part of the update cycle.

This problem requires *composable learning*, the capability for models learned by different systems to be directly composed as a single model that combines the strengths of each component. We need model representations that can be joined without additional training, which is a major challenge for modern data-driven (deep) machine learning. Different agents with different experiences will invariably develop different representations that cannot be directly combined. Though the agents may use the same deep neural network architecture, each structurally-equivalent weight will be specialized for different purposes.

To capture this notion of composability, we choose instead to use models that live in a Reproducing Kernel Hilbert Space (RKHS), a classic non-parametric learning approach, in which a learned model is represented directly over its training data [149]. Because the domain of the learned model's applicability is directly represented by its support, we can imagine how models that were developed over separate regions of the data

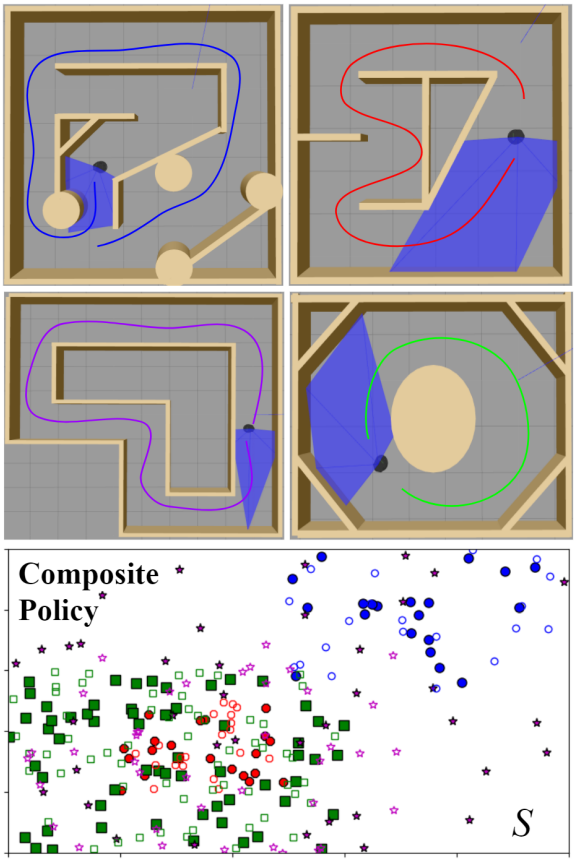


Figure 7.1. Four policies are trained on individual environments and each represented with their own kernel support and then composed into a single policy with broader support that applies everywhere. These four training environments (referred to as Maze, Circuit 1, Circuit 2, and Round) were used in simulation as part of the demonstration of our approach to composable learning.

space can combine their supports into one broader model. Recent work on online learning of sparse kernel functions [122] gives us tools for building larger kernel functions while minimizing the number of basis points required in the support.

We apply this model composition approach to the scenario of robots learning to avoid obstacles by experiencing collisions, a well-studied problem that has been successfully solved on physical platforms using methods that blend reinforcement learning with foundational techniques in optimal control [150] or supervised learning [151]. Because our primary interest is understanding the composability of obstacle-avoiding policies, we opt to learn our policies using Q-learning as a stand-in for more sophisticated approaches, based on recent work in Q-learning with sparse kernel representations [152]. To enable learning control policies in the continuous action space of our problem, we propose a kernelized version of the Normalized Advantage Function Q-learning algorithm [153].

In the context of Q-learning, the density of kernel support of the converged policy is a reflection of the states where we can expect that policy to be near optimal, since these states are drawn from the stationary distribution of the Bellman optimality operator. We suggest the use of the density of the support to decide which component policy should be chosen to dominate in a composition of policies in a region of the state space. We can represent this density directly in the RKHS as an application of the Kernel Mean Embedding [154], but with the addition of our sparsity-inducing procedures.

The **main contributions** of this work are: 1) An algorithm for composing multiple RKHS functions according to their density of kernel support; 2) A Kernel Normalized Advantage Function Q-learning algorithm that extends [153] to learn policies represented in an RKHS; 3) A proof-of-concept demonstration in which we: (a) train obstacle-avoidance policies in multiple simulations of a robot equipped with a laser scanner and navigating in a 2D environment; (b) apply composition to various policy combinations and test them to show that the composed policies retain the performance of their components; and (c) transfer the composed policy directly to a physical platform operating in an arena with obstacles in order to elicit a sense of generalization.

This chapter has been published as [155]. An open-source implementation of our work can be found at `https://github.com/katetolstaya/kernelrl`. A video demonstration is available at `https://youtu.be/kWIigy5MWdU`.

A Normalized Advantage Functions in RKHS

We model an autonomous agent in a continuous space as a Markov Decision Process (MDP) with continuous states $\mathbf{s} \in \mathcal{S} \subseteq \mathbb{R}^p$ and actions $\mathbf{a} \in \mathcal{A} \subseteq \mathbb{R}^q$. When in state $\mathbf{s}$ and taking action $\mathbf{a}$, a random transition to state $\mathbf{s}'$ occurs according to the conditional probability density $\mathbb{P}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$. After the agent to a particular $\mathbf{s}'$ from $\mathbf{s}$, the MDP assigns an instantaneous reward $r(\mathbf{s}, \mathbf{a}, \mathbf{s}')$, where the reward function is a map $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$.

In Markov Decision problems[98], the goal is to find the action sequence $\{\mathbf{a}_t\}_{t=0}^{\infty}$ so as to maximize the infinite horizon accumulation of rewards, i.e., the value: $V(\mathbf{s}, \{\mathbf{a}_t\}_{t=0}^{\infty}) := \mathbb{E}_{\mathbf{s}'}[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t) \mid \mathbf{s}_0 = \mathbf{s}, \{\mathbf{a}_t\}_{t=0}^{\infty}]$ [129]. The action-value function $Q(\mathbf{s}, \mathbf{a})$ is the conditional mean of the value function given the initial action $\mathbf{a}_0 = \mathbf{a}$:

$$Q\left(\mathbf{s}, \mathbf{a}, \{\mathbf{a}_t\}_{t=1}^{\infty}\right) := \mathbb{E}_{\mathbf{s}'}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t) \mid \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}, \{\mathbf{a}_t\}_{t=1}^{\infty}\right]$$

We consider the case where actions $\mathbf{a}_t$ are chosen according to a stationary stochastic policy, where a policy is a mapping from states to actions: π: $\mathcal{S} \to \mathcal{A}$. We define $Q^*(\mathbf{s}, \mathbf{a})$ as the maximum of (A) with respect to the action sequence.

It is possible to formulate finding the optimal action-value function as a fixed point problem [156]: shift the index of the summand in (A) by one, make use of the time invariance of the Markov transition kernel, and the homogeneity of the summand, to derive the Bellman optimality equation :

$$Q^*(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}'}\left[r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}')\right] \tag{7.1}$$

The reason for defining action-value functions is that the optimal Q^* may be used to compute the optimal policy π^* as

$$\pi^*(\mathbf{s}) = \operatorname*{argmax}_{\mathbf{a}} Q^*(\mathbf{s}, \mathbf{a}) \ . \tag{7.2}$$

Computation of the optimal policy for continuous action problems requires maximizing the Q function (7.2) which may not have a closed form, and can be challenging. To mitigate this issue, we hypothesize that

Q has a special form: it is a sum of the value and an *advantage* [157]:

$$Q(\mathbf{s}, \mathbf{a}) = V(\mathbf{s}) + A(\mathbf{s}, \mathbf{a}) \tag{7.3}$$

According to the definition of the action-value function, we require that $\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}) = V(\mathbf{s})$. Therefore, the optimal action has no advantage: $\max_{\mathbf{a}} A^*(\mathbf{s}, \mathbf{a}) = 0$.

For this hypothesis to yield computational savings, we parametrize the advantage function as a quadratic function [153]. We define $L(\mathbf{s}) : \mathcal{S} \to \mathcal{A} \times \mathcal{A}$ as a matrix function, so $L^T(\mathbf{s})L(\mathbf{s})$ is positive definite. The policy is a bounded vector function $\pi : \mathcal{S} \to \mathcal{A}$.

$$A(\mathbf{s}, \mathbf{a}) = -\frac{1}{2} \left(\mathbf{a} - \pi(\mathbf{s})\right) L^T(\mathbf{s}) L(\mathbf{s}) \left(\mathbf{a} - \pi(\mathbf{s})\right) \tag{7.4}$$

Thus, we assume the advantage function is quadratic in the actions, which is restrictive, albeit sufficient, for the applications we examine.

To find the optimal policy, we seek to satisfy (7.1) for all state-action pairs, yielding the cost functional $\tilde{J}(Q)$:

$$\tilde{J}(V, \pi, L) = \mathbb{E}_{\mathbf{s},\mathbf{a}} \left(y(\mathbf{s}, \mathbf{a}) - Q(\mathbf{s}, \mathbf{a})\right)^2, \tag{7.5}$$

where $y(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}'}[r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma V(\mathbf{s}')]$. Substituting (7.3), finding the Bellman fixed point reduces to the stochastic program:

$$V^*, L^*, \pi^* = \underset{V, \pi, L \in \mathcal{B}(\mathcal{S})}{\operatorname{argmin}} \; \tilde{J}(V, \pi, L) \; . \tag{7.6}$$

Since searching over value functions that are arbitrary bounded continuous functions $\mathcal{B}(\mathcal{S})$ is untenable, we restrict $\mathcal{B}(\mathcal{S})$ to be a reproducing Kernel Hilbert space (RKHS) $\mathcal{H}$ to which $V(\mathbf{s})$ belongs [129]. Further, the quadratic parameterization of the advantage function means that π and L also belong to $\mathcal{H}$. An RKHS over $\mathcal{S}$ is a Hilbert space is equipped with a reproducing kernel, an inner product-like map $\kappa : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ [133, 134]:

$$\text{(i)} \quad \langle \pi, \kappa(\mathbf{s}, \cdot) \rangle_{\mathcal{H}} = \pi(\mathbf{s}), \qquad \text{(ii)} \quad \mathcal{H} = \operatorname{span}\{\kappa(\mathbf{s}, \cdot)\} \tag{7.7}$$

In this work, we restrict the kernel used to be in the family of universal kernels, such as a Gaussian kernel with constant diagonal covariance Σ, i.e.,

$$\kappa(\mathbf{s}, \mathbf{s}') = \exp\left\{-\frac{1}{2}(\mathbf{s}-\mathbf{s}')\,\Sigma\,(\mathbf{s}-\mathbf{s}')^T\right\} \tag{7.8}$$

motivated by the fact that a continuous function over a compact set may be approximated uniformly by a function in a RKHS equipped with a universal kernel [136].

To apply the Representer Theorem, we require the cost to be coercive in V, π and L [134], which may be satisfied through use of a Hilbert-norm regularizer, so we define the regularized cost functional $J(V, \pi, L) = \tilde{J}(V, \pi, L) + (\lambda/2)(\|V\|_{\mathcal{H}}^2 + \|\pi\|_{\mathcal{H}}^2 + \|L\|_{\mathcal{H}}^2)$ and solve the regularized problem.

$$\begin{aligned} V^*, \pi^*, L^* &= \operatorname*{argmin}_{V,\pi,L\in\mathcal{H}} J(V, \pi, L) \\ &= \operatorname*{argmin}_{V,\pi,L\in\mathcal{H}} \tilde{J}(V, \pi, L) + \frac{\lambda}{2}\left(\|V\|_{\mathcal{H}}^2 + \|\pi\|_{\mathcal{H}}^2 + \|L\|_{\mathcal{H}}^2\right). \end{aligned} \tag{7.9}$$

We point out that this is a step back from the original intent of solving (7.6) because the assumption we have made about V^*, π^*, and L^* being representable in the RKHS $\mathcal{H}$ need not be true. More importantly, the functional $J(V, \pi, L)$ is not convex in V, π, and L and there is no guarantee that a local minimum of $J(V, \pi, L)$ will be the optimal policy π^*. This is a significant difference relative to policy evaluation problems [117]. In the next section, we describe a method for solving (7.9) in parallel and then composing sparse solutions.

B Composable Learning in RKHS

To enable efficient learning in diverse environments in parallel, we require a composable representation of the control policy $\pi(\mathbf{s})$, which is obtained through a sparse kernel parametrization. First, we develop an algorithm for learning these policy representations using a sequence of observations from a single environment.

Q-Learning with Kernel Normalized Advantage Functions. To solve (7.9), we follow the semi-gradient TD approach described by [101], which uses the directional derivative of the loss along the direction where $\hat{y}_t(\mathbf{s}, \mathbf{a})$ is fixed and independent of Q_t. To obtain the semi-gradient at the sample $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}'_t)$, we define the fixed target value $y_t(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}'_t) = r_t + \gamma V_t(\mathbf{s}'_t)$ and the temporal difference as $\delta_t = y_t - Q_t(\mathbf{s}_t, \mathbf{a}_t)$. Then, we

apply the chain rule to the resulting functional stochastic directional derivative, together with the reproducing property of the RKHS (7.7) to obtain the stochastic functional semi-gradients of the loss $J(V, \pi, L)$ as

$$\begin{aligned}
\hat{\nabla}_V J(V, \pi, L) &= -\delta_t \kappa(\mathbf{s}_t) \\
\hat{\nabla}_\pi J(V, \pi, L) &= -L(\mathbf{s}_t) L(\mathbf{s}_t)^T \left(\mathbf{a}_t - \pi_t(\mathbf{s}_t)\right) \delta_t \kappa(\mathbf{s}_t) \\
\hat{\nabla}_L J(V, \pi, L) &= L(\mathbf{s}_t)^T \left(\mathbf{a}_t - \pi_t(\mathbf{s}_t)\right) \left(\mathbf{a}_t - \pi_t(\mathbf{s}_t)\right)^T \delta_t \kappa(\mathbf{s}_t)
\end{aligned} \tag{7.10}$$

As a result, $V, \pi, L \in \mathcal{H}$ are expansions of kernel evaluations only at past observed states and the optimal V, π and L functions in the Reproducing Kernel Hilbert Space (RKHS) take the following form

$$V(\mathbf{s}) = \sum_{n=1}^{N} w_{Vn} \kappa(\mathbf{s}_n, \mathbf{s}), \quad \pi(\mathbf{s}) = \sum_{n=1}^{N} \mathbf{w}_{\pi n} \kappa(\mathbf{s}_n, \mathbf{s}), \quad L(\mathbf{s}) = \sum_{n=1}^{N} \mathbf{w}_{Ln} \kappa(\mathbf{s}_n, \mathbf{s}) \tag{7.11}$$

We propose the Kernel Normalized Advantage Functions (KNAF) variant of Q-Learning to iteratively learn the action-value function while following trajectories with a stochastic policy. This algorithm includes the approximation of a kernel density metric $\rho(\mathbf{s})$, which is later used to compose multiple learned models (7.13). The V, π, L and ρ function representations are compressed with Kernel Orthogonal Matching Pursuit (KOMP), where we tie the compression to the learning rate to preserve tractability and convergence of the learning process [120, 122].

Alg. 8 produces locally optimal policy representations of low complexity, sparsified using KOMP. These parsimonious representations reduce the complexity of real-time policy evaluation and enable the efficient policy composition procedure described in the next section.

Model Composition. The key novelty of our approach is the ability to compose multiple control policies without additional training samples. We summarize a procedure for projecting N policies trained in parallel on different environments using Algorithm 8 $\pi_i(\mathbf{s}) = \sum_j w_{ij} \kappa(\mathbf{s}, \mathbf{s}_{ij})$,for $i = 1, \ldots, N$, to a single function $\Pi(\mathbf{s}) = \sum_j w_j \kappa(\mathbf{s}, \mathbf{s}_j)$.

We seek an interpolation between multiple candidate policies, which preserves the values of the original functions. A simple linear combination or sum is insufficient because we want to set $\Pi(\mathbf{s}) = \pi_i(\mathbf{s})$ for all $\mathbf{s}$ in the kernel dictionaries of the candidate policies. To project multiple functions π_i into one RKHS, we iterate

Algorithm 8 Q-Learning with Kernel Normalized Advantage Functions (KNAF)

Require: l_0, $\{\alpha_t, \beta_t, \zeta_t, \epsilon_t, \Sigma_t\}_{t=0,1,2...}$

$V_0(\cdot) = 0, \pi_0(\cdot) = 0, L_0(\cdot) = l_0 I, \rho_0(\cdot) = 0$

for $t = 0, 1, 2, \ldots$ **do**

Obtain trajectory realization $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}'_t)$ using a stochastic policy $\pi_t(\mathbf{s}_t) \sim \mathcal{N}(\pi_t(\mathbf{s}_t), \Sigma_t)$

Compute the target value and Bellman error

$$\hat{y}_t(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}'_t) = r_t + \gamma V_t(\mathbf{s}'_t), \quad \delta_t = \hat{y}_t - Q_t(\mathbf{s}_t, \mathbf{a}_t)$$

Compute the stochastic estimates of the gradients of J with respect to V, π and L

$$\hat{\nabla}_V J(Q_t) = -\delta_t \kappa(\mathbf{s}_t)$$

$$\hat{\nabla}_\pi J(Q_t) = -L(\mathbf{s}_t) L(\mathbf{s}_t)^T (\mathbf{a}_t - \pi_t(\mathbf{s}_t)) \delta_t \kappa(\mathbf{s}_t)$$

$$\hat{\nabla}_L J(Q_t) = L(\mathbf{s}_t)^T (\mathbf{a}_t - \pi_t(\mathbf{s}_t))(\mathbf{a}_t - \pi_t(\mathbf{s}_t))^T \delta_t \kappa(\mathbf{s}_t)$$

Update V, π, L, ρ:

$$V_{t+1} = V_t(\cdot) - \alpha_t \hat{\nabla}_V J(Q_t), \quad \pi_{t+1} = \pi_t(\cdot) - \beta_t \hat{\nabla}_\pi J(Q_t),$$

$$L_{t+1} = L_t(\cdot) - \zeta_t \hat{\nabla}_L J(Q_t) \quad , \rho_{t+1} = \rho_t(\cdot) + \kappa(\mathbf{s}_t)$$

Obtain greedy compression of V_{t+1}, π_{t+1}, L_{t+1}, ρ_{t+1} via KOMP with budget ϵ_t

end for

return V,π,L

through all dictionary points and perform the following update on $\Pi(\cdot)$ at $\mathbf{s}_{ij}$ given $\pi_i(\mathbf{s}_{ij})$:

$$\Pi(\cdot) = \Pi(\cdot) + \big(\pi_i(\mathbf{s}_{ij}) - \Pi(\mathbf{s}_{ij})\big)\,\kappa(\mathbf{s}_{ij}, \cdot) \tag{7.12}$$

The next challenge we address is resolving local conflicts among π_i. We score the reliability of the π_i in the neighborhood of $\mathbf{s}$ by the number of gradient steps performed in that neighborhood during training, which is the same as the number of observations at that state in Alg. 8. We cannot directly use the density of kernel dictionary elements because the pruning step of Alg. 8 limits the density of the representation via compression with budget ϵ_t. To accurately represent the number of observations in a neighborhood of the state space, we propose to augment the approximation of V, π and L with a fourth function, ρ, which is the kernel mean embedding of the observed states, representing the probability distribution of states observed during training [158]. To incorporate this approach into Algorithm 8, we augment Step 6 of Algorithm 8 with a density estimation step:

$$\rho_{t+1} = \rho_t(\cdot) + \kappa(\mathbf{s}_t, \cdot) \tag{7.13}$$

When ρ is repeatedly pruned along with V, π, and L, the removed weights are projected onto nearby dictionary elements, producing non-unity weights for each dictionary element. This results in a sparse representation of the kernel mean embedding, accurate to ϵ over the entire state-space. This approach is inspired by kernel density estimation [154].

In Algorithm 9, we first accumulate dictionary points from all candidate policies into one matrix D and initialize the composite policy to zero, $\Pi(\cdot) = 0$. Then, we choose points $\mathbf{s}_{ij}$ from D uniformly without replacement and compare the kernel density at $\mathbf{s}_{ij}$ of π_i, the function of origin for that observation, against the density at $\mathbf{s}_{ij}$ of π_k for $k \neq i$, the other candidate functions. If the density of π_i is greater, we add that point to the composite result because it is deemed most reliable in the neighborhood of $\mathbf{s}_{ij}$ in the state space.

For our application, we are able to estimate ρ indirectly by the density of dictionary points of a policy π around state $\mathbf{s}$:

$$\tilde{\rho}(\pi, \mathbf{s}) = \sum_{\mathbf{s}_k \in \pi} \kappa(\mathbf{s}, \mathbf{s}_k) \tag{7.14}$$

This simplification can be used because the structure of the reward function induces an a approximation of a kernel mean embedding on the kernel dictionary of π itself. This does not hold for all problems. For example,

Algorithm 9 Composition with Conflict Resolution

Input: $\{\pi_i(\mathbf{s}) = \sum_j^{M_i} w_{ij}\kappa(\mathbf{s}, \mathbf{s}_{ij}), \quad \rho_i(\mathbf{s}) = \sum_j^{M_i} v_{ij}\kappa(\mathbf{s}, \mathbf{s}_{ij})\}_{i=1,2\ldots,N}, \epsilon$

1: Initialize $\Pi(\cdot) = 0$, append centers $D = [\mathbf{s}_{11}, \ldots, \mathbf{s}_{ij}, \ldots]$
2: **for** each $\mathbf{s}_{ij} \in D$ chosen uniformly at random **do**
3: **if** $\rho_i(\mathbf{s}_{ij}) > \max_{k \neq i} \rho_k(\mathbf{s}_{ij})$ **then**
4:

$$\Pi = \Pi(\cdot) + \left(\pi_i(\mathbf{s}_{ij}) - \Pi(\mathbf{s}_{ij})\right) \kappa(\mathbf{s}_{ij}, \cdot)$$

5: **end if**
6: **end for**
7: Obtain compression of π using KOMP with ϵ
8: **return** f

if the optimal value, policy, and L functions are all zero in a region of the state space, then all kernel points in the region will be pruned away and the density of the dictionary will not accurately represent the reliability of the function.

C Simulated & Experimental Evaluations

We apply the KNAF algorithm to the robotic obstacle avoidance task by training on a variety of environments in simulation, and then validating the learned policies on a physical robot.

Simulation Results Building on the infrastructure designed by [159], we train our algorithm on a wheeled robot traveling through indoor environments. The robot receives laser scans at a rate of 10 Hz, with 5 range readings at at an angular interval of 34° with a field of view of 170°. The robot controls its angular velocity between $[-0.3, 0.3]$ rad/s at a rate of 10Hz while traveling with a constant forward velocity 0.15 m/s. While traveling through the environment, the robot receives a reward of -200 for colliding with obstacles and a reward of +1 otherwise. The four training environments pictured in Fig. 7.1 are Maze, Circuit 1, Circuit 2 and Round, which mimic the appearance of indoor spaces. The Round environment is simplest of the four environments because of its radial symmetry. The Circuit 2 environment incorporates narrow hallways that turn both left and right. The Circuit 1 environment adds multiple tight turns in quick succession. The Maze environment incorporates all of these features in a more complex environment.

Individual environment training. We demonstrate the typical learning progress of Algorithm 8 by reproducing ten trials of training with the Round environment, and plot training rewards, model order, and

Bellman error in Fig. 7.2. The learning progress is extremely reproducible, with average episode rewards of at least 2000 after 100,000 training steps. By the end of training, the robot travels for at least 2000 steps before crashing during every episode. Training Bellman error reliably converges for every trial. The resulting policies are parsimonious, with a limiting model order of fewer than 250 kernel dictionary elements. Next, we demonstrate the performance of Alg 8 on the four environments pictured in Fig. 7.1 in Table 7.1. All policies achieve zero crashes during testing. We also observe that the Maze environment required many more training steps and achieved a larger model order. We interpret this to mean that the Maze environment is more varied, and therefore requires a higher coverage of the state space.

Generalization across environments. We analyze the inability of a single policy to generalize to other environments in Table 7.2. We observe that policies trained on the Round environment (Policy 1) could not generalize to other environments at all, with an average test reward of -4226. We observe that all policies were able to successfully navigate the Round environment with no crashes, most likely due to its simplicity. The policy trained on the Maze environment (Policy 2) was able to somewhat generalize to the other three environments, while none of the other environments could generalize to the Maze, due to its complexity. Policies trained on Circuit 1 (Policy 4) and Circuit 2 (Policy 3) completely failed on the Maze environment, but were able to transfer somewhat among each other, and to the Round environment.

Dual compositions. We apply the composition algorithm (Alg. 9) to combine two of the four trained policies to observe performance on the original two environments and analyze generalization performance on the remaining environments in Fig. 7.2. All controllers composed from Policy 2 (trained on Maze) achieved perfect reward of 1000 on the difficult Maze environment. Policies composed from Circuit 1 and Circuit 2 were able to achieve positive average rewards on the Circuit 1 and 2 environments. We observe some generalization capabilities in the results for the composite Policy 1/2 (Round and Maze), which performs better in all four test environments than both of the original policies. The other dual composite policies were not able to generalize better than the individual policies.

Composition of multiple policies. We further validate the composition algorithm (Alg. 9) using every combination of four trained policies: an additional eleven policies composed from two, three, and four of the original four policies. The performance of the composite policies was validated on the four training environments in Fig. 7.2. Nearly optimal performance was observed when all four policies (1/2/3/4) were composed into one, and then tested on all of the four environments. Two collisions were observed in the

Environment	Steps	Model Order	Loss	Rewards
Round	230K	224	2.24	**1000**
Maze	630K	779	16.04	**1000**
Circuit 2	280K	467	36.40	**1000**
Circuit 1	500K	578	47.97	**1000**

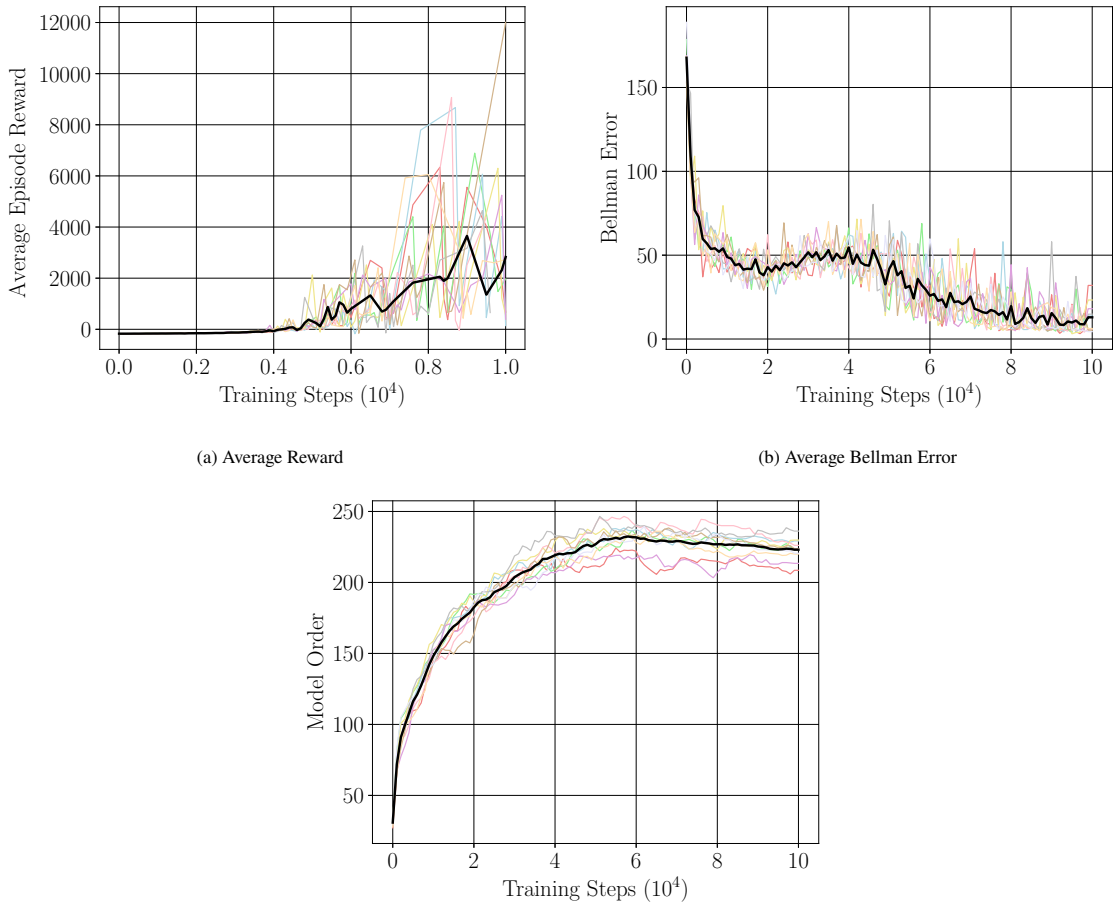
Table 7.1: Limiting model complexity, training loss (Bellman error), and accumulation of rewards over 1,000 testing steps for each of the four environments. All environments are solved by the K-NAF algorithm within 700,000 simulation steps.

Circuit 1 environment, and none in the other three when testing for 1,000 time steps in each environment.

Robot Results. The real-world validation experiments were carried out on a Scarab robot pictured in Fig. 7.3, equipped with an on-board computer, wireless communication, and a Hokuyo URG laser range finder. It is actuated by stepper motors and its physical dimensions are 30 x 28 x 20 cm with a mass of 8kg [160]. Laser scans were received at a rate of 10Hz and 5 range readings were obtained as in simulation. Angular velocity commands were issued at 10Hz. The test environment was built using laboratory furniture and miscellaneous equipment pictured in Fig. 7.4. When testing the policy trained only on the Round environment, we observed 3 collisions, with a total reward of 397 over 1,000 time steps. Using the composite policy, which incorporates all 4 policies, no collisions were observed during 1,000 time steps. In Fig. 7.5, we visualize the trajectory obtained by testing the composite policy on the Scarab robot, which shows that the robot successfully completed multiple loops around the obstacles in the environment.

D Summary

We envision multi-robot systems which can exchange concise representations of their knowledge to enable distributed learning and control in complex environments with limited communication [7]. In this work, we take the first step towards this goal by developing and demonstrating the Q-Learning with KNAF algorithm, which allows the composition of multiple learned policies without additional training samples. Rather than a single robot collecting information about multiple environments in sequence, multiple robots could work in parallel and combine their models. We also see a clear path to extend this approach to higher-dimensional problems using an auto-encoder to learn policies in an RKHS based on a feature space representation, rather than directly in the state space [150].



(a) Average Reward

(b) Average Bellman Error

(c) Model Order of Q

Figure 7.2. Results of 10 experiments over 100, 000 training steps were averaged (black curve) to demonstrate the learning progress for the robotic obstacle avoidance task with the Round environment. Fig. 7.2a shows the average reward obtained by the stochastic policy during training shows that the robot was able to complete 5000 simulation steps without crashing by the end of training. Fig. 7.2b shows the Bellman error for training samples converges to a small non-zero value. The model order of the Q approximation remains under 200 for all ten experiments. The exploration variance Σ was 0.2. Constant learning rates were used, $\alpha_t = 0.25$, $\beta_t = 0.25$, $\zeta_t = 0.001$. L was initialized to $L_0 = 0.01I$. For the KOMP Parameters, we used a Gaussian kernel with a bandwidth of $[0.75, 0.75, 0.75, 0.75, 0.75]$ and a pruning tolerance of $\epsilon_t = 3.0$.

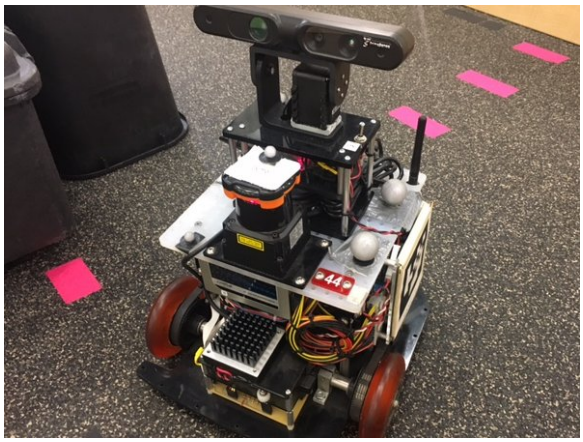


Figure 7.3. Scarab robot with equipped with an on-board computer, wireless communication, and a Hokuyo URG laser range finder.

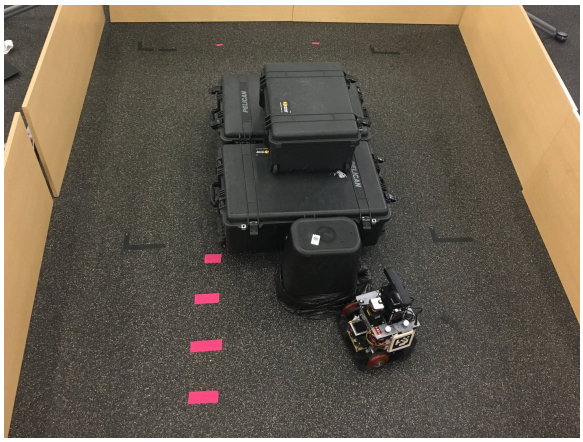


Figure 7.4. We validate our approach by testing policies trained in simulation on a real robot in a laboratory environment. The policy trained only on the Round environment experienced 3 crashes over 1,000 testing steps. The composite 1/2/3/4 policy in Table 7.2, combined from all four policies, received a reward of 1,000 during 1,000 testing steps, with no crashes.

Policies / Reward	Round	Maze	Circuit 2	Circuit 1
1 - Round	**1000**	-11663	-608	-608
2 - Maze	**1000**	**1000**	-5	-407
3 - Circuit 2	**1000**	-11663	**1000**	196
4 - Circuit 1	**1000**	-11462	-407	**1000**
1 / 2	**1000**	**1000**	-5	-206
1 / 3	**1000**	-11663	799	-206
1 / 4	**1000**	-11261	-206	799
2 / 3	**1000**	**1000**	**1000**	-5
2 / 4	**1000**	**1000**	-5	799
3 / 4	**1000**	-11462	397	397
1 / 2 / 3	**1000**	**1000**	799	196
1 / 2 / 4	**1000**	**1000**	-5	**1000**
1 / 3 / 4	**1000**	-11663	397	799
2 / 3 / 4	**1000**	**1000**	799	-206
1 / 2 / 3 / 4	**1000**	**1000**	**1000**	598

Table 7.2: Cross-validation was performed on all possible compositions of the original 4 policies, each of which was trained on one environment pictured in Fig. 7.1. Each row is a policy composed using Alg 9 of one or more policies trained using Algorithm 8. All 15 policies were tested on the 4 environments, and compared based on the reward accumulated over 1,000 time steps in each environment.

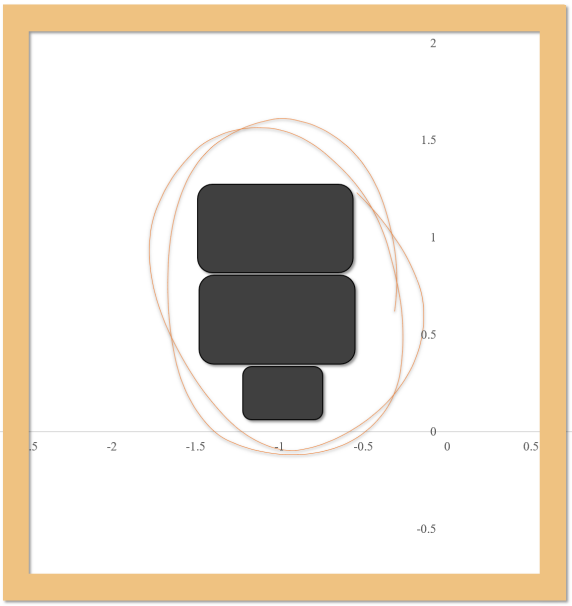


Figure 7.5. We visualize the trajectory observed when testing the composite policy (Policy 1/2/3/4) in Table 7.2 on the Scarab robot. Position coordinates were computed through onboard motor odometry, which results in a small drift in the position.

Chapter 8

Conclusions and Future Work

The use of large-scale robot teams could significantly improve the speed of autonomous mapping, infrastructure inspection, agricultural monitoring, and search and rescue operations. We have developed a set of algorithmic tools to allow cooperative, autonomous robot systems to scale to dozens of robots. We address the key scalability issues in the existing algorithms for coordination, control, and communication by designing machine learning abstractions that leverage graph neural networks and scale efficiently to complex tasks and large teams of robots. We conclude the thesis by proposing an approach for distributed learning, in which robots train controllers independently and then compose their representations.

In this dissertation, all controllers were trained in simulation. To deploy these approaches to physical systems, we anticipate the need for transfer learning from simulation to hardware. The systems are likely to see to a change the distribution of the observations, due to stochasticity in the control rates and communication delays of physical systems, requiring re-training or fine-tuning on the physical hardware. While the low-level collision avoidance controllers may need to be retrained, we envision that the high-level coordination policies for tasks, such as coverage or exploration, could be transferred zero-shot to the real world. In addition, the physical implementation of ad-hoc networks may require the development of new low cost networking hardware and protocols. Also, depending on the hardware used by the robots, the model architecture and the implementation of graph operations may need to be optimized for real-time inference.

This dissertation also assumes that the teams of robots are homogeneous, with the same capabilities for control, sensing and communication. The extension of our approach to heterogeneous teams would require

the encoding of diverse robot capabilities as part of the team's graph. The heterogeneous team may also include human team members, which would require the design of human-robot interfaces for visualizing the robots' local information, and updating mission specifications.

Finally, we envision the extension of the GNN methodology to systems of non-cooperative agents. An adversarial task could require resilience within the team, in which robots need to react to loss of team members. Existing approaches typically require rounds of communication among all team members [161], which may be impractical in large teams. Most situations are not purely cooperative or adversarial. Autonomous driving is an example of a task in which agents may have competing objectives, but must cooperate for safety. Behavior prediction of human drivers is a key problem for autonomous driving in urban environments, and GNNs have already been successfully deployed to encode relationships between agents on a road [162].

Bibliography

[1] FlightAware. (2019) Seattle-tacoma international airport. [Online]. Available: https://flightaware.com/live/airport_status_bigmap.rvt?airport=KSEA

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[4] V. Sharma, M. Bennis, and R. Kumar, "UAV-assisted heterogeneous networks for capacity enhancement," *IEEE Communications Letters*, vol. 20, no. 6, pp. 1207–1210, 2016.

[5] S. Thrun, W. Burgard, and D. Fox, "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping," in *Robotics and Automation, 2000. Proceedings. ICRA 2000. IEEE International Conference on*, vol. 1. IEEE, 2000, pp. 321–328.

[6] S. Thrun and Y. Liu, "Multi-robot slam with sparse extended information filers," in *Robotics Research. The Eleventh International Symposium*. Springer, 2005, pp. 254–266.

[7] B. Schlotfeldt, D. Thakur, N. Atanasov, V. Kumar, and G. J. Pappas, "Anytime planning for decentralized multirobot active information gathering," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1025–1032, 2018.

[8] J. L. Baxter, E. Burke, J. M. Garibaldi, and M. Norman, "Multi-robot search and rescue: A potential field based approach," in *Autonomous robots and agents*. Springer, 2007, pp. 9–16.

[9] J. S. Jennings, G. Whelan, and W. F. Evans, "Cooperative search and rescue with a team of mobile robots," in *Advanced Robotics, 1997. ICAR'97. Proceedings., 8th International Conference on*. IEEE, 1997, pp. 193–200.

[10] H. Zhang and J. C. Hou, "Maintaining sensing coverage and connectivity in large sensor networks," *Ad Hoc & Sensor Wireless Networks*, vol. 1, no. 1-2, pp. 89–124, 2005.

[11] H. S. Witsenhausen, "A counterexample in stochastic optimum control," *SIAM Journal on Control*, vol. 6, no. 1, pp. 131–147, 1968.

[12] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro, "Convolutional neural network architectures for signals supported on graphs," *IEEE Trans. Signal Process.*, vol. 67, no. 4, pp. 1034–1049, Feb. 2019.

[13] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[14] B. Chen, B. Dai, and L. Song, "Learning to plan via neural exploration-exploitation trees," *arXiv preprint arXiv:1903.00070*, 2019.

[15] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," *arXiv preprint arXiv:1906.01227*, 2019.

[16] F. Chen, S. Bai, T. Shan, and B. Englot, "Self-learning exploration and mapping for mobile robots via deep reinforcement learning," in *AIAA Scitech 2019 Forum*, 2019, p. 0396.

[17] J. Paulos, S. W. Chen, D. Shishika, and K. Vijay, "Decentralization of multiagent policies by learning what to communicate," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, May 2019.

[18] M. Hüttenrauch, A. Šošić, and G. Neumann, "Deep reinforcement learning for swarm systems," *Journal of Machine Learning Research*, vol. 20, no. 54, pp. 1–31, 2019.

[19] G. A. D. Caro, A. Giusti, J. Nagi, and L. M. Gambardella, "A simple and efficient approach for cooperative incremental learning in robot swarms," in *2013 16th International Conference on Advanced Robotics (ICAR)*, Nov. 2013, pp. 1–8.

[20] A. Giusti, J. Nagi, L. Gambardella, and G. A. D. Caro, "Cooperative sensing and recognition by a swarm of mobile robots," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, pp. 551–558.

[21] M. Otte, "An emergent group mind across a swarm of robots: Collective cognition and distributed sensing via a shared wireless neural network," *The International Journal of Robotics Research*, vol. 37, no. 9, pp. 1017–1061, Aug. 2018.

[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th Int. Conf. Learning Representations*. Toulon, France: Assoc. Comput. Linguistics, 24-26 Apr. 2017.

[23] K. Zhou, J. C. Doyle, K. Glover *et al.*, *Robust and optimal control*. Prentice hall New Jersey, 1996, vol. 40.

[24] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[25] C. Eksin, P. Molavi, A. Ribeiro, and A. Jadbabaie, "Bayesian quadratic network game filters," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 4589–4593.

[26] A. Sandryhaila and J. M. F. Moura, "Big data analysis with signal processing on graphs," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 80–90, Sep. 2014.

[27] S. Chen, R. Varma, A. Sandryhaila, and J. Kovačević, "Discrete signal processing on graphs: Sampling theory," *IEEE Trans. Signal Process.*, vol. 63, no. 24, pp. 6510–6523, Dec. 2015.

[28] A. G. Marques, S. Segarra, G. Leus, and A. Ribeiro, "Sampling of graph signals with successive local aggregations," *IEEE Trans. Signal Process.*, vol. 64, no. 7, pp. 1832–1843, Apr. 2016.

[29] F. Gama, J. Bruna, and A. Ribeiro, "Stability properties of graph neural networks," *IEEE Transactions on Signal Processing*, vol. 68, pp. 5680–5695, 2020.

[30] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro, "Learning decentralized controllers for robot swarms with graph neural networks," in *Conference on Robot Learning*. PMLR, 2020, pp. 671–682.

[31] E. Tolstaya, J. Paulos, V. Kumar, and A. Ribeiro, "Multi-robot coverage and exploration using spatial graph neural networks," *arXiv preprint arXiv:2011.01119*, 2020.

[32] F. Gama, A. G. Marques, A. Ribeiro, and G. Leus, "Aggregation graph neural networks," in *44th IEEE Int. Conf. Acoust., Speech and Signal Process.* Brighton, UK: IEEE, 12-17 May 2019.

[33] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous systems*, vol. 61, no. 12, pp. 1258–1276, 2013.

[34] A. Khan, E. Tolstaya, A. Ribeiro, and V. Kumar, "Graph policy gradients for large scale robot control," in *Conference on Robot Learning*. PMLR, 2020, pp. 823–834.

[35] P. Toth and D. Vigo, *The vehicle routing problem*. SIAM, 2002.

[36] L. Perron and V. Furnon. OR-Tools. Google. [Online]. Available: https://developers.google.com/optimization/

[37] Q. Sykora, M. Ren, and R. Urtasun, "Multi-agent routing value iteration network," *Int. Conf. on Machine Learning (ICML)*, 2020.

[38] M. McNaughton, C. Urmson, J. M. Dolan, and J.-W. Lee, "Motion planning for autonomous driving with a conformal spatiotemporal lattice," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 4889–4895.

[39] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the operations research society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[40] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[41] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[43] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, 1989, pp. 305–313.

[44] A. Giusti, J. Guzzi, D. C. Ciresan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots." *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2016.

[45] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations (ICLR)*, Banff, Apr. 2014. [Online]. Available: http://arxiv.org/abs/1213.6203

[46] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Annu. Conf. Neural Inform. Process. Syst. 2016*. Barcelona, Spain: NIPS Foundation, 5-10 Dec. 2016.

[47] L. Ruiz, F. Gama, A. G. Marques, and A. Ribeiro, "Median activation functions for graph neural networks," in *44th IEEE Int. Conf. Acoust., Speech and Signal Process.* Brighton, UK: IEEE, 12-17 May 2019. [Online]. Available: http://arxiv.org/abs/1810.12165

[48] A. Tacchetti, H. F. Song, P. A. M. Mediano, V. Zambaldi, J. Kramar, N. C. Rabinowitz, T. Graepel, M. Botvinick, and P. W. Battaglia, "Relational forward models for multi-agent learning," in *International Conference on Learning Representations*, New Orleans, May 2019.

[49] H. G. Tanner, "Flocking with obstacle avoidance in switching networks of interconnected vehicles," in *IEEE International Conference on Robotics and Automation*, vol. 3. Citeseer, 2004, pp. 3006–3011.

[50] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 25–34.

[51] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on automatic control*, vol. 48, no. 6, pp. 988–1001, 2003.

[52] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Stable flocking of mobile agents, Part II: dynamic topology," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 2. IEEE, 2003, pp. 2016–2021.

[53] L. Xiao, S. Boyd, and S.-J. Kim, "Distributed average consensus with least-mean-square deviation," *Journal of parallel and distributed computing*, vol. 67, no. 1, pp. 33–46, 2007.

[54] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.

[55] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017.

[56] D. Mellinger, "Trajectory generation and control for quadrotors," Ph.D., University of Pennsylvania, Philadelphia, 2012.

[57] M. Prandini, L. Piroddi, S. Puechmorel, and S. L. Brázdilová, "Toward air traffic complexity assessment in new generation air traffic management systems," *IEEE transactions on intelligent transportation systems*, vol. 12, no. 3, pp. 809–818, 2011.

[58] Z. Mahboubi and M. J. Kochenderfer, "Autonomous air traffic control for non-towered airports," in *Proc. USA/Eur. Air Traffic Manage. Res. Develop. Seminar*, 2015, pp. 1–6.

[59] K. Tumer and A. Agogino, "Distributed agent-based air traffic flow management," in *Proceedings of the 6th international joint Conf. on Autonomous agents and multiagent systems*. ACM, 2007, p. 255.

[60] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proceedings. 1991 IEEE Int. Conf. on Robotics and Automation*. IEEE, 1991, pp. 1398–1404.

[61] P. Vadakkepat, K. C. Tan, and W. Ming-Liang, "Evolutionary artificial potential fields and their application in real time robot path planning," in *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, vol. 1, July 2000, pp. 256–263 vol.1.

[62] K. Ramamoorthy, T. Singh, and J. Crassidis, "Potential functions for en-route air traffic management and flight planning," in *AIAA Guidance, Navigation, and Control Conf. and Exhibit*, 2004, p. 4878.

[63] X. Xu, C. Li, and Y. Zhao, "Air traffic rerouting planning based on the improved artificial potential field model," in *Control and Decision Conf. (CCDC), 2010 Chinese*. IEEE, 2010, pp. 1444–1449.

[64] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Towards search-based motion planning for micro aerial vehicles," *arXiv preprint arXiv:1810.03071*, 2018.

[65] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Computation*, vol. 3, no. 1, pp. 88–97, 1991.

[66] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of the 23rd Int. Conf. on Machine learning*. ACM, 2006, pp. 729–736.

[67] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first Int. Conf. on Machine learning*. ACM, 2004, p. 1.

[68] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *AAAI*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[69] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," *arXiv preprint arXiv:1507.04888*, 2015.

[70] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *Int. Conf. on Machine Learning*, 2016, pp. 49–58.

[71] E. Tolstaya, A. Ribeiro, V. Kumar, and A. Kapoor, "Inverse optimal planning for air traffic control," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 7535–7542.

[72] H. Chitsaz and S. M. LaValle, "Time-optimal paths for a Dubins airplane," in *Decision and Control, 2007 46th IEEE Conf. on*. IEEE, 2007, pp. 2379–2384.

[73] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ Int. Conf. on*. IEEE, 2017, pp. 2872–2879.

[74] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in neural information processing systems*, 2004, pp. 767–774.

[75] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[76] E. T. Jaynes, "Information theory and statistical mechanics," *Physical review*, vol. 106, no. 4, p. 620, 1957.

[77] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal, "Learning objective functions for manipulation," in *2013 IEEE Int. Conf. on Robotics and Automation*. IEEE, 2013, pp. 1331–1336.

[78] R. Murphy, R. R. Murphy, and R. C. Arkin, *Introduction to AI robotics*. MIT press, 2000.

[79] F. A. Administration. (2016) Terminal procedures publication. [Online]. Available: https://www.faa.gov/air_traffic/flight_info/aeronav/productcatalog/ifrcharts/terminalprocedures/

[80] J. Farrell and M. Barth, *The global positioning system and inertial navigation*. Mcgraw-hill New York, 1999, vol. 61.

[81] G. van Drimmelen. (2018) GPS Utils. [Online]. Available: https://gist.github.com/govert/1b373696c9a27ff4c72a

[82] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-based motion planning for aggressive flight in SE(3)," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2439–2446, 2018.

[83] S. Community. (2019) Univariate spline. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.UnivariateSpline.html

[84] B. Williams and T. Camp, "Comparison of broadcasting techniques for mobile ad hoc networks," in *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, 2002, pp. 194–205.

[85] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu, "The broadcast storm problem in a mobile ad hoc networks," *Wireless networks*, vol. 8, no. 2, pp. 153–167, 2002.

[86] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016, pp. 2252–2260.

[87] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016, pp. 2137–2145.

[88] A. Singh, T. Jain, and S. Sukhbaatar, "Learning when to communicate at scale in multiagent cooperative and competitive tasks," in *International Conference on Learning Representations*, 2018.

[89] J. P. Inala, Y. Yang, J. Paulos, Y. Pu, O. Bastani, V. Kumar, M. Rinard, and A. Solar-Lezama, "Neurosymbolic transformers for multi-agent communication," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[90] Y.-C. Liu, J. Tian, N. Glaser, and Z. Kira, "When2com: Multi-agent perception via communication graph grouping," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 4106–4115.

[91] E. Tolstaya, L. Butler, D. Mox, J. Paulos, V. Kumar, and A. Ribeiro, "Learning connectivity for data distribution in robot teams," *arXiv preprint arXiv:2103.05091*, 2021.

[92] A. Goldsmith, *Wireless communications*. Cambridge university press, 2005.

[93] U. Challita and W. Saad, "Network formation in the sky: Unmanned aerial vehicles for multi-hop wireless backhauling," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2017, pp. 1–6.

[94] A. Al-Hourani, S. Kandeepan, and A. Jamalipour, "Modeling air-to-ground path loss for low altitude platforms in urban environments," in *2014 IEEE Global Communications Conference*, 2014, pp. 2898–2904.

[95] R. Shorey, A. Ananda, M. C. Chan, and W. T. Ooi, *Mobile, wireless, and sensor networks: technology, applications, and future directions*. John Wiley & Sons, 2006.

[96] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol *et al.*, "Stable baselines," 2018.

[97] G. Miao, J. Zander, K. W. Sung, and S. B. Slimane, *Fundamentals of mobile data networks*. Cambridge University Press, 2016.

[98] R. Bellman, "The theory of dynamic programming," DTIC Document, Tech. Rep., 1954.

[99] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, p. 0278364913495721, 2013.

[100] D. P. Bertsekas and S. E. Shreve, *Stochastic optimal control: The discrete time case*. New York: Academic Press, 1978.

[101] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 2018.

[102] K. Mitkovska-Trendova, R. Minovski, and D. Boshkovski, "Methodology for transition probabilities determination in a Markov decision processes model for quality-accuracy management," *Journal of Engineering Management and Competitiveness (JEMC)*, vol. 4, no. 2, pp. 59–67, 2014.

[103] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[104] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[105] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, UK, May 1989.

[106] W. B. Powell and J. Ma, "A review of stochastic algorithms with continuous value function approximation and some new approximate policy iteration algorithms for multidimensional continuous applications," *Journal of Control Theory and Applications*, vol. 9, no. 3, pp. 336–352, 2011.

[107] R. S. Sutton, H. R. Maei, and C. Szepesvári, "A convergent $O(n)$ temporal-difference algorithm for off-policy learning with linear function approximation," in *Advances in neural information processing systems*, 2009, pp. 1609–1616.

[108] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.

[109] J. N. Tsitsiklis, "Asynchronous stochastic approximation and Q-learning," *Machine Learning*, vol. 16, no. 3, pp. 185–202, 1994.

[110] T. Jaakkola, M. I. Jordan, and S. P. Singh, "On the convergence of stochastic iterative dynamic programming algorithms," *Neural computation*, vol. 6, no. 6, pp. 1185–1201, 1994.

[111] S. Bhatnagar, D. Precup, D. Silver, R. S. Sutton, H. R. Maei, and C. Szepesvári, "Convergent temporal-difference learning with arbitrary smooth function approximation," in *Advances in Neural Information Processing Systems*, 2009, pp. 1204–1212.

[112] G. Kimeldorf and G. Wahba, "Some results on Tchebycheffian spline functions," *Journal of mathematical analysis and applications*, vol. 33, no. 1, pp. 82–95, 1971.

[113] L. Baird, "Residual algorithms: Reinforcement learning with function approximation," in *In Proceedings of the Twelfth International Conference on Machine Learning*. Morgan Kaufmann, 1995, pp. 30–37.

[114] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE transactions on automatic control*, vol. 42, no. 5, pp. 674–690, 1997.

[115] N. K. Jong and P. Stone, "Model-based function approximation in reinforcement learning," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM, 2007, p. 95.

[116] Y. Ermoliev, "Stochastic quasigradient methods and their application to system optimization," *Stochastics: An International Journal of Probability and Stochastic Processes*, vol. 9, no. 1-2, pp. 1–36, 1983.

[117] A. Koppel, G. Warnell, E. Stump, P. Stone, and A. Ribeiro, "Breaking Bellman's curse of dimensionality: Efficient kernel gradient temporal difference," *arXiv preprint arXiv:1709.04221 (Submitted to TAC Dec. 2017)*, 2017.

[118] S. Grünewälder, G. Lever, L. Baldassarre, M. Pontil, and A. Gretton, "Modelling transition dynamics in MDPs with RKHS embeddings," in *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, vol. 1, 2012, pp. 535–542.

[119] B. Dai, N. He, Y. Pan, B. Boots, and L. Song, "Learning from conditional distributions via dual embeddings," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1458–1467.

[120] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on signal processing*, vol. 41, no. 12, pp. 3397–3415, 1993.

[121] G. Lever, J. Shawe-Taylor, R. Stafford, and C. Szepesvari, "Compressed conditional mean embeddings for model-based reinforcement learning," in *30th AAAI Conference on Artificial Intelligence*, 2016.

[122] A. Koppel, G. Warnell, E. Stump, and A. Ribeiro, "Parsimonious online learning with kernels via sparse projections in function space," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 83–126, 2019.

[123] E. J. Candes, "The restricted isometry property and its implications for compressed sensing," *Comptes Rendus Mathematique*, vol. 346, no. 9, pp. 589–592, 2008.

[124] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online learning with kernels," in *Advances in neural information processing systems*, 2002, pp. 785–792.

[125] M. Wang, E. X. Fang, and H. Liu, "Stochastic compositional gradient descent: algorithms for minimizing compositions of expected-value functions," *Mathematical Programming*, vol. 161, no. 1-2, pp. 419–449, 2017.

[126] A. W. Moore, "Efficient memory-based learning for robot control," University of Cambridge, Computer Laboratory, Tech. Rep., Nov. 1990.

[127] K. Yoshida, "Swing-up control of an inverted pendulum by energy-based methods," in *Am. Control Conf.*, vol. 6. IEEE, 1999, pp. 4045–4047.

[128] Z. Wang, K. Crammer, and S. Vucetic, "Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale SVM training," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 3103–3131, 2012.

[129] E. Tolstaya, A. Koppel, E. Stump, and A. Ribeiro, "Nonparametric stochastic compositional gradient descent for Q-learning in continuous Markov decision problems," in *2018 Annual American Control Conference (ACC)*. IEEE, 2018, pp. 6608–6615.

[130] A. Koppel, E. Tolstaya, E. Stump, and A. Ribeiro, "Nonparametric stochastic compositional gradient descent for Q-learning in continuous Markov decision problems," *arXiv preprint arXiv:1804.07323*, 2018.

[131] S. Chen, A. M. Devraj, A. Bušić, and S. Meyn, "Zap Q-learning with nonlinear function approximation," *arXiv preprint arXiv:1910.05405*, 2019.

[132] R. Wheeden, R. Wheeden, and A. Zygmund, *Measure and Integral: An Introduction to Real Analysis*, ser. Chapman & Hall/CRC Pure and Applied Math. Taylor & Francis, 1977.

[133] V. Norkin and M. Keyzer, "On stochastic optimization and statistical learning in reproducing kernel Hilbert spaces by support vector machines (SVM)," *Informatica*, vol. 20, no. 2, pp. 273–292, 2009.

[134] A. Argyriou, C. A. Micchelli, and M. Pontil, "When is there a representer theorem? Vector versus matrix regularizers," *Journal of Machine Learning Research*, vol. 10, no. Nov, pp. 2507–2529, 2009.

[135] Y. Nesterov, "Smooth minimization of non-smooth functions," *Math. Program.*, vol. 103, no. 1, pp. 127–152, 2005.

[136] C. A. Micchelli, Y. Xu, and H. Zhang, "Universal kernels," *Journal of Machine Learning Research*, vol. 7, no. Dec, pp. 2651–2667, 2006.

[137] J. Friedman, T. Hastie, R. Tibshirani *et al.*, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1, no. 10.

[138] S. B. Gelfand and S. K. Mitter, "Recursive stochastic algorithms for global optimization in R^d," *SIAM J. Control Optim.*, vol. 29, no. 5, pp. 999–1018, 1991.

[139] V. R. Konda and V. S. Borkar, "Actor-critic–type learning algorithms for Markov decision processes," *SIAM J. Control Optim.*, vol. 38, no. 1, pp. 94–123, 1999.

[140] V. S. Borkar, "Stochastic approximation with two time scales," *Syst. Control Lett.*, vol. 29, no. 5, pp. 291–294, 1997.

[141] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Transactions on signal processing*, vol. 52, no. 8, pp. 2275–2285, 2004.

[142] P. Vincent and Y. Bengio, "Kernel matching pursuit," *Machine Learning*, vol. 48, no. 1, pp. 165–187, 2002.

[143] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvari, "Convergence results for single-step on-policy reinforcement-learning algorithms," *Machine learning*, vol. 38, no. 3, pp. 287–308, 2000.

[144] M. Wang and D. Bertsekas, "Incremental constraint projection-proximal methods for nonsmooth convex optimization," *SIAM J. Optim.*, 2014.

[145] OpenAI Gym - continuous mountain car. [Online]. Available: gym.openai.com/envs/MountainCarContinuous-v0/

[146] F. S. Melo and M. I. Ribeiro, "Q-learning with linear function approximation," in *International Conference on Computational Learning Theory*. Springer, 2007, pp. 308–322.

[147] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*. PMLR, 2014, pp. 387–395.

[148] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[149] B. Schölkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[150] C. Richter and N. Roy, "Safe visual navigation via deep learning and novelty detection," in *Proc. of the Robotics: Science and Systems Conference*, 2017.

[151] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[152] E. Tolstaya, A. Koppel, E. Stump, and A. Ribeiro, "Nonparametric stochastic compositional gradient descent for Q-learning in continuous Markov decision problems," in *American Control Conference*, 2018.

[153] S. Gu, T. P. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-learning with model-based acceleration," *CoRR*, vol. abs/1603.00748, 2016.

[154] K. Muandet, K. Fukumizu, B. Sriperumbudur, B. Schölkopf *et al.*, "Kernel mean embedding of distributions: A review and beyond," *Foundations and Trends in Machine Learning*, vol. 10, no. 1-2, pp. 1–141, 2017.

[155] E. Tolstaya, E. Stump, A. Koppel, and A. Ribeiro, "Composable learning with sparse kernel representations," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 4622–4628.

[156] D. P. Bertsekas and S. Shreve, *Stochastic optimal control: the discrete-time case*. Academic Press, 2004.

[157] L. C. Baird, "Reinforcement learning in continuous time: Advantage updating," in *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, vol. 4. IEEE, 1994, pp. 2448–2453.

[158] S. J. Sheather and M. C. Jones, "A reliable data-based bandwidth selection method for kernel density estimation," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 683–690, 1991.

[159] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, "Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and gazebo," *arXiv preprint arXiv:1608.05742*, 2016.

[160] N. Atanasov, J. Le Ny, N. Michael, and G. J. Pappas, "Stochastic source seeking in complex environments," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 3013–3018.

[161] B. Schlotfeldt, V. Tzoumas, D. Thakur, and G. J. Pappas, "Resilient active information gathering with mobile robots," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4309–4316.

[162] S. Casas, C. Gulino, R. Liao, and R. Urtasun, “Spatially-aware graph neural networks for relational behavior forecasting from sensor data,” *arXiv preprint arXiv:1910.08233*, 2019.