2021

Graph Convolutions For Teams Of Robots

Arbaaz Khan
University of Pennsylvania

Graph Convolutions For Teams Of Robots

Abstract

In many applications in robotics, there exist teams of robots operating in dynamic environments requiring the design of complex communication and control schemes. The problem is made easier if one assumes the presence of an oracle that has instantaneous access to states of all entities in the environment and can communicate simultaneously without any loss. However, such an assumption is unrealistic especially when there exist a large number of robots. More specifically, we are interested in decentralized control policies for teams of robots using only local communication and sensory information to achieve high level team objectives. We first make the case for using distributed reinforcement learning to learn local behaviours by optimizing for a sparse team wide reward as opposed to existing model based methods. A central caveat of learning policies using model free reinforcement learning is the lack of scalability. To achieve large scale scalable results, we introduce a novel paradigm where the policies are parametrized by graph convolutions. Additionally, we also develop new methodologies to train these policies and derive technical insights into their behaviors. Building upon these, we design perception action loops for teams of robots that rely only on noisy visual sensors, a learned history state and local information from nearby robots to achieve complex team wide-objectives. We demonstrate the effectiveness of our methods on several large scale multi-robot tasks.

Degree Type

Dissertation

Degree Name

Doctor of Philosophy (PhD)

Graduate Group

Electrical & Systems Engineering

First Advisor

Vijay Kumar

Second Advisor

Alejandro Ribeiro

Keywords

Graph Convolutions, Machine Learning, Multi-Agent, Reinforcment Learning, Robotics, Swarm

Subject Categories

Computer Sciences | Robotics

GRAPH CONVOLUTIONS FOR TEAMS OF ROBOTS

Arbaaz Khan

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

Supervisor of Dissertation

_____

Alejandro Ribeiro,
Professor,
Electrical and Systems Engineering

Co-Supervisor of Dissertation

_____

Vijay Kumar,
Professor,
Electrical and Systems Engineering

Graduate Group Chairperson

Alejandro Ribeiro, Professor of Electrical and Systems Engineering

Dissertation Committee

Pratik Chaudhari, Assistant Professor, Electrical and Systems Engineering

Aleksandra Faust, Ph.D Computer Science

GRAPH CONVOLUTIONS FOR TEAMS OF ROBOTS

If I have seen further, it is by standing on the shoulders of giants.

Sir Isaac Newton

ACKNOWLEDGMENTS

No work stands alone and this thesis is no exception. I would like to begin by offering my thanks to every teacher I ever had, be it in a formal educational setting or otherwise. Without them this work would not exist today. I have been extremely lucky to have the privilege of working with some extraordinarily talented people over the last five years at Penn.

I would like to start by thanking my primary advisor Alejandro Ribeiro who not only has an astute ability to devise solutions for seemingly intractable complex research problems but also has infinite reserves of patience to hear out my arguments. I would also like to thank my other primary advisor Vijay Kumar for making me appreciate the bigger research picture and for instilling an appreciation for espresso. I am without doubt a better researcher and a better person thanks to my advisors and for this I shall forever be indebted to them. I would also like to extend my gratitude to my commmittee chair Pratik Chaudhari for his help, support and encouragement navigating the complex landscape of research and academia. I am also hugely indebted to Aleksandra Faust who not only introduced me to cutting edge research at Google but has also time and again offered her support and wisdom.

The list of people who have helped me out on this journey are many, but I would like to thank as many of them as possible

- I would like to begin with thanking my mother for her constant willingness to listen to my rambling at the oddest of hours. Her encouragement, love and support has

been hugely monumental in completing this body of work. I would also like to thank my father for his love, support and willingness to remind me time and again what is important in life. If not for my parents, this work would not be complete.

- I would also like to thank my academic brother-in-arms, roommate and friend Clark Zhang. I will forever cherish our trips to conferences, our chess games, late night conversations about research and life.
- Pooja for putting up with all my princely airs while writing any paper or this thesis. Your support has meant everything to me.
- Brent for those trips to Atlantic City where we pondered upon the intricacies of human psychology and Kate for teaching me how to ski despite me not wanting to. You made my time at Penn that much more enjoyable.
- Sarah Tang who guided me through the rigours of academia like an elder sibling would.
- Nikolai at NVIDIA, Daniel Lee at Samsung Research, Anthony Francis at Google Brain and Stefan at Apple. Your wisdom and guidance has helped me immensely in being a better researcher.
- Last but not the least, Luna for being the purfect furry companion!

ABSTRACT

Graph Convolutions for Teams of Robots

Arbaaz Khan

Alejandro Ribeiro

Vijay Kumar

In many applications in robotics, there exist teams of robots operating in dynamic environments requiring the design of complex communication and control schemes. The problem is made easier if one assumes the presence of an oracle that has instantaneous access to states of all entities in the environment and can communicate simultaneously without any loss. However, such an assumption is unrealistic especially when there exist a large number of robots. More specifically, we are interested in decentralized control policies for teams of robots using only local communication and sensory information to achieve high level team objectives. We first make the case for using distributed reinforcement learning to learn local behaviours by optimizing for a sparse team wide reward as opposed to existing model based methods. A central caveat of learning policies using model free reinforcement learning is the lack of scalability. To achieve large scale scalable results, we introduce a novel paradigm where the policies are parametrized by graph convolutions. Additionally, we also develop new methodologies to train these policies and derive technical insights into their behaviors. Building upon these, we design perception action loops for teams of robots that rely only on noisy visual sensors, a learned history state and local information from nearby robots to achieve complex team wide-objectives. We demonstrate the effectiveness of our methods on several large scale multi-robot tasks.

CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1 INTRODUCTION

In a world met with an increasing demand for automation and robotics, many applications require teams of robots to collaborate in order to complete a task who's complexity would far exceed the operational capabilities of any individual robot in the team. Examples of such tasks include but are not limited to formation flying [1, 2], warehouse management with teams of robots [3], multi-robot furniture assembly [4], concurrent control and communication for teams of robots [5], perimeter defense and surveillance [6].

In order to meaningfully solve real world multi-robot tasks, we would like our potential solutions to be decentralized and scalable. In a system where there are many robots operating simultaneously, it is almost infeasible for each robot to know where all the other robots and entities in the environment are at any given time. Even if one were to set aside this communication constraint for a moment and assume that a paradigm exists which enables lossless and instantaneous communication between all robots. However, now we run into the curse of dimensionality. As the size of the robot team grows, each robot must instantaneously digest a large amount of information, produce a meaningful control action and broadcast relevant information for other robots.

In the past traditional solutions to solving multi-robot focused on minimizing/maximizing for a carefully setup optimization problem constrained by robot and environment dynamics. A key drawback of such hand designed cost functions can be that adding a simple additional constraint or increasing the number of agents can render the optimization problem intractable [7]. For example, in labeled multi-agent planning scenarios,

such as warehouse management [3] or delivery [8], each robot must navigate to a fixed, non-interchangeable goal state, presumably to complete a given sub task. Unfortunately, the curse of dimensionality causes most planning algorithms to become intractable for large robot teams; in fact, even planning for two-dimensional disk-shaped robots in this setting has been shown to be a strongly NP-hard problem [9].

In the recent past, deep learning has proved to be an extremely valuable tool for robotics. Harnessing the power of deep neural networks has emerged as a successful approach to designing policies that map sensor inputs to control outputs for complex tasks. These include, but are not limited to, learning to play video games [10, 11], learning complex control policies for robot tasks [12] and learning to plan in unknown environments with only sensor information [13–15]. However, most of these works are limited to a single robot or single entity and often require the learning algorithm to ingest large amounts of training data. It naturally follows that directly applying these learning algorithms to multi-robot problems would require larger datasets and longer training times as the number of robots increases. In fact, later in Chapter 2 we show that the amount of training data/training time required for a simple multi-robot problem grows exponentially as the number of robots in the team increase. How then do we go about leveraging machine learning to solve some of these problems?

The key idea explored in this work is to design intelligent machine learning solutions that look to leverage the underlying structure of the problem in order to achieve scalability and decentralized control for teams of robots. This work proposes new advances in the fields of robotics and machine learning that looks to leverage the local symmetry and structure naturally occurring in teams of robots to learn compute local control policies that optimize for a global cost. This thesis can be broadly divided into three sections;

- **Machine Learning for Teams of Robots** We first start by looking to answer why one should consider using using model free learning based solutions for multi-robot

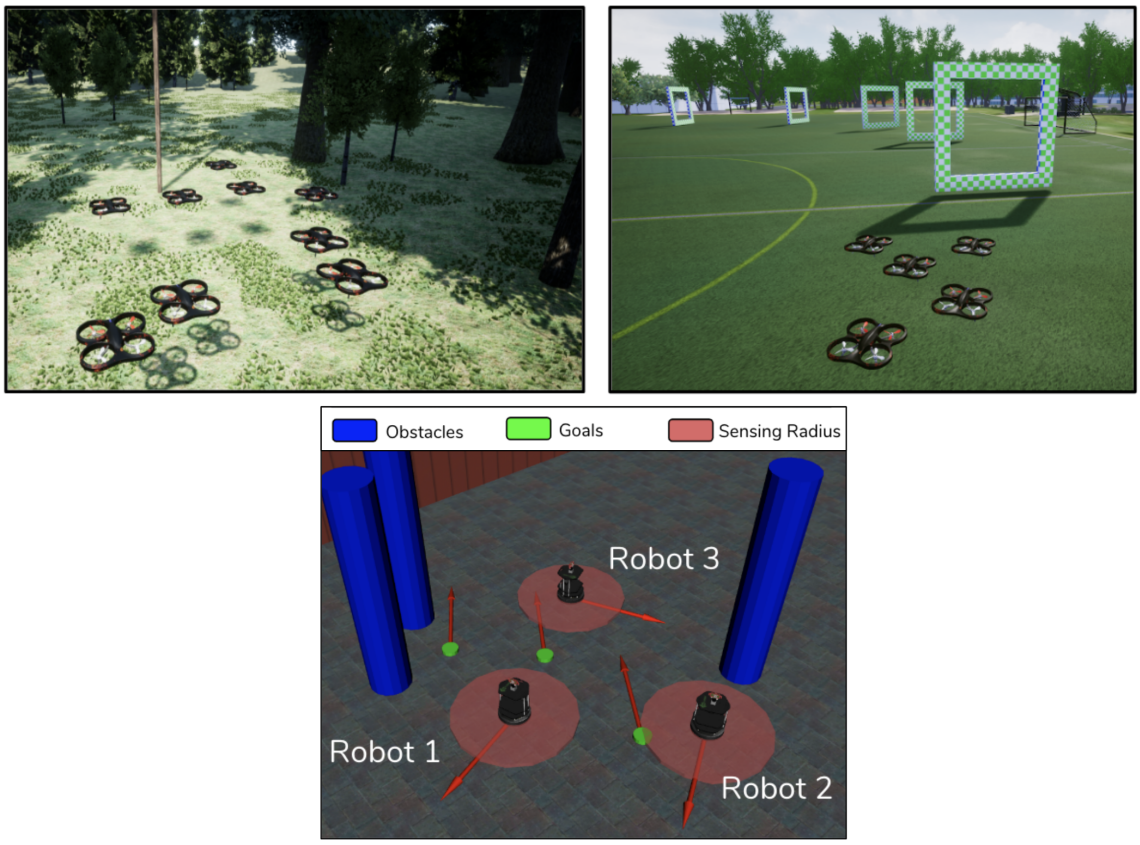


Figure 1.1: **Examples of multi-robot problems requiring large scale decentralized solutions (Top Left)** A team of robots navigating through a cluttered outdoor environment where each roboy must rely only on its own sensors and information coming from nearby robots to navigate through the environment without collisions and also staying within a certain distance to the other robots. **(Top Right)** A team of robots flying through a series of checkpoints sequenced in a specific order in the least amount of time possible. The robots do not have information about the location of the checkpoints beforehand and must decide on control actions on the fly to achieve team wide objectives. **(Bottom)** Unlabeled motion planning where there exist N robots and N goals but it doesn't matter which robot goes to which goal as long as all goals are covered.

problems. To answer this question, we analyze the classic problem of unlabeled motion planning where there exist N identical robots and N goals, but it doesn't matter which robot navigates to which goal. This is a well studied problem in robotics and there exist many model based solutions that either propose a centralized solution or a decentralized solution by relaxing constraints robot dynamics, obstacles, start or

goal orientations. We demonstrate that using off the shelf multi-agent reinforcement learning to learn decentralized control policies guided by a sparse and central team wide function can offer several benefits over existing model based solutions. These ideas are explored in depth in Chapter 2

- **Graph Convolutions for Robot Teams** A key caveat of using off the shelf reinforcement solutions for multi-robot problems is the inability of these solutions to scale with the number of robots or entities in the environment. To overcome these, we propose a new policy parametrization paradigm using graph convolutions that inherently looks to not just compute policies, but also learn to communicate with the right subset of neighbors. We also develop a new algorithm to effectively train these policies such that at inference time, the solutions can be scaled up to a large number of robots. In addition to detailed simulations to verify the effectiveness of our proposed graph convolution based solutions, we also derive key theoretical results. These ideas are explored in Chapter 3 and Chapter 4.

- **Graph Memory Convolutions for Perception Action Communication Loops** We build on our graph convolutions for multi-robot teams by moving away from our earlier simplifying assumptions of perfect information about the environment. We develop solutions to learn what we call perception-action-communication loops for teams of robots operating in complex environments but only equipped with simple visual sensors and simple communication capabilities. Additionally, we also take a step away from our earlier simplistic reactive controllers towards more complex control schemes required when robots are operating in complex environments and need to achieve multiple objectives. To do so, we propose yet another paradigm where our graph convolutions are not just defined over a robots neighbors but also over a robot's belief state and its neighbors belief states. We demonstrate the efficacy

of our methods by considering varied multi-robot problems in challenging scenarios. These ideas are discussed in Chapter 5 and Chapter 6.

We conclude this thesis with still open problems and possible future directions in Chapter 7.

2 REINFORCEMENT LEARNING FOR TEAMS OF ROBOTS

In many applications in robotics such as formation flying [2, 1] or perimeter defense and surveillance [6], there exist teams of interchangeable robots operating in complex environments. In these scenarios, the goal is to have a team of robots execute a set of identical tasks such that each robot executes only one task, but it does not matter which robot executes which task. One example of such a problem is the concurrent goal assignment and trajectory planning problem where robots must simultaneously assign goals and plan motion primitives to reach assigned goals.

Solutions to this unlabeled multi-robot planning problem must solve both the goal assignment and trajectory optimization problems. It has been shown that the flexibility to freely assign goals to robots allows for polynomial-time solutions under certain conditions [16–18]. Nonetheless, there are still significant drawbacks to existing approaches. Solutions with polynomial-time complexities depend on minimum separations between start positions, goal positions, or robots and obstacles [16, 17, 19]. Motion plans generated by graph-based approaches for this problem such as those proposed in [18, 20] are limited to simple real-world robots that have approximately first-order dynamics. Closest to our work, [21] proposes an algorithm to coordinate unlabeled robots with arbitrary dynamics in obstacle-filled environments. However, it assumes the existence of a single-robot trajectory optimizer that can produce a candidate trajectory for a robot to any given goal, which is in itself a difficult research problem. Furthermore, the algorithm is a priority-based method that depends on offsetting the times at which robots start traversing their trajectories to

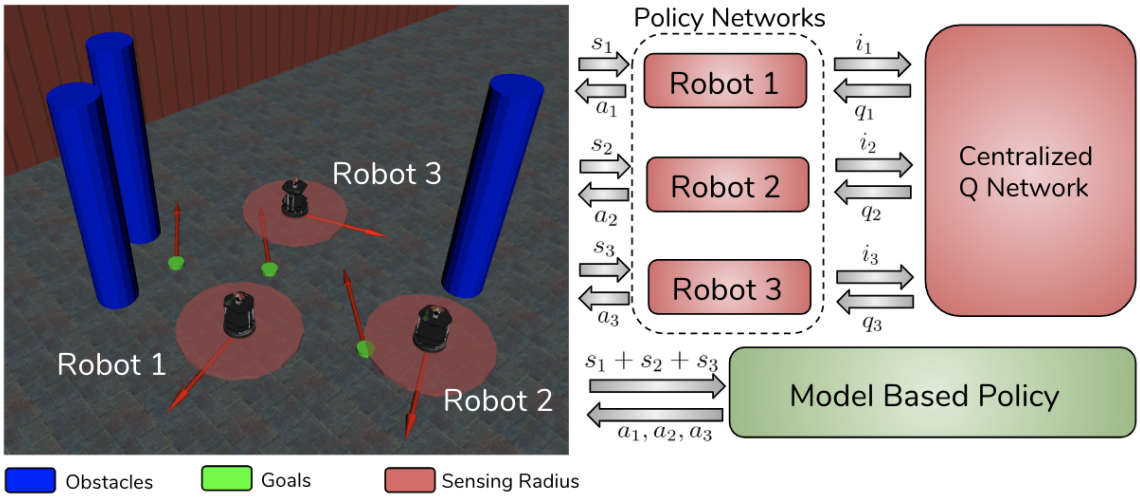


Figure 2.1: Learning Unlabeled Motion Planning Robots observe their own state and velocity, relative positions of all goals and other entities within a sensing region. For robot n, this information is compiled into a vector s_n. Each robot uses its own policy network to compute an action a_n. During training, policy networks also exchange information i with a centralized Q-network which uses information from all robots to compute robot specific Q functions q_n. These Q functions are used by the individual robot functions to update their policies. To guarantee safety, a model based policy runs in the background.

guarantee collision avoidance. In the worst case, it degenerates into a completely sequential algorithm where only one robot is moving at a time.

In light of these works, we propose a novel learning-based framework for goal assignment and trajectory optimization for a team of identical robots with arbitrary dynamics operating in obstacle-filled work spaces. Firstly, it is observed that the unlabeled multi-robot planning problem can be recast as a multi-agent reinforcement learning (MARL) problem. Robots are given their own state, configuration of obstacles and information about other robots within some sensing radius and configuration of all goals in the environment. The objective then is to learn policies that couple the assignment and trajectory generation for each robot. It is important to note that in such an approach, since we do not have goals assigned beforehand, it is non-trivial to assign individual rewards to each robot. Instead, we simply assign a global reward to all robots that takes into account if all robots have

reached goals in a collision free manner. This global reward then forces each robot to learn policies such that all robots can reach a goal without colliding or having to communicate with each other. Thus, by casting the concurrent goal assignment problem as a MARL problem, we attempt to learn policies that maximize this global reward in minimal time. A centralized training, decentralized execution strategy is used to train policies for each robot. This builds on a body of work in MARL that employ such a centralized training, decentralized execution strategy [22, 23].

When utilizing a deep reinforcement learning (RL) model, one loses optimality guarantees as well as any guarantees for collision free trajectories. To ensure one still has collision free trajectories, we make use of an analytical model based policy that runs in the background and checks if the target velocities produced by the robot are "safe". For each robot, velocity obstacles are computed at every instant in time. The velocity obstacle divides the set of all possible velocities into safe and unsafe sets. If the velocity computed by the learned model lies in the unsafe set, it is projected back into the safe set. We also ensure that this projection preserves certain properties such as smoothness of the transition function in order to be compatible with the learning process. Thus, when using this model based policy in conjunction with the policy learned, we are guaranteed almost safe trajectories at every instant and it is empirically shown that the computed policies converge to the desired values.

Thus, the main contributions of our algorithm are : 1) Capability to extend to arbitrary robot dynamics. We test with both holonomic and non-holonomic robot models 2) There is no need to re-train if the obstacles are moved around or if the robot's start and goal positions are changed. This is in contrast to any model-based approach where one would need to recompute the assignment and regenerate the trajectory. 3) It retains the safety guarantees of model-based methods, 4) We show experimentally that better performance based on total time to reach goals is achieved than model based methods alone.

2.1 BACKGROUND AND PROBLEM FORMULATION

Consider a two dimensional Euclidean space with N homogeneous disk-shaped robots of radius R indexed by n and M goal locations indexed by m. Goal location m is represented as a vector of its position (x_m, y_m) and heading (θ_m) in the plane. Thus, goal m is :

$$\mathbf{g}_m = [x_m, y_m, \theta_m] \tag{2.1}$$

The full goal state vector, $\mathbf{G} \in \mathbb{SE}(2)^M$ is given as :

$$\mathbf{G} = [\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_M] \tag{2.2}$$

Let robot n be equipped with a sensor with range R_s ($R_s > R$). The data from the sensor is denoted as I_n. We assume each robot has full information about the location of the goals and full information about its own pose. Thus, the observation of the nth robot at time t is then given as :

$$\mathbf{o}_t^n = [\mathbf{p}_n(t), v_n(t), \omega_n(t), I_n, \mathbf{G}] \tag{2.3}$$

where $\mathbf{p}_n(t) \in \mathbb{SE}(2)$ is a vector of the position $(x_n(t), y_n(t))$ and heading $(\theta_n(t))$ of robot n in the plane and is given as :

$$\mathbf{p}_n(t) = [x_n(t), y_n(t), \theta_n(t)] \tag{2.4}$$

The linear velocity of the robot at time t is denoted as $v_n(t)$ and the angular velocity is denoted as $\omega_n(t)$. Let $\mathcal{S}$ be the set of states describing all possible configurations of all robots. For each robot, given an action $a_t^n \in \mathcal{A}_n$ where $\mathcal{A}_n$ describes all possible actions for robot n, the state of the robot evolves according to some stationary dynamics

distribution with conditional density $p(\mathbf{o}_{t+1}|\mathbf{o}_t, a_t)$. These dynamics can be linear or non-linear. Further let $\mathcal{A} = [\mathcal{A}_1, \mathcal{A}_2, \ldots \mathcal{A}_n]$ be the set of all possible actions of all robot. We define some arbitrary scalar $\delta > 0$ such that the necessary and sufficient condition to ensure collision avoidance is given as :

$$\begin{gathered} E_c(p_i(t), p_j(t)) > 2R + \delta, \\ \text{for all} i \neq j \in \{1, \ldots N\}, \text{for all} t \end{gathered} \tag{2.5}$$

where E_c is the euclidean distance. Lastly, we define the assignment matrix $\phi(t) \in \mathbb{R}^{N \times M}$ as

$$\phi_{ij}(t) = \begin{cases} 1, & \text{if } E_c(p_i(t), g_j) + D_c(p_i(t), g_j) \leqslant \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

where D_C is the cosine distance and ϵ is some threshold. In this work, we consider the case when $N = M$. The necessary and sufficient condition for all goals to be covered by robots at some time $t = T$ is then:

$$\phi(T)^\top \phi(T) = \mathbf{I}_N \tag{2.7}$$

where $\mathbf{I}$ is the identity matrix. Thus, the concurrent assignment and planning problem statement considered here can be defined as :

***Problem* 2.1.1:** Given an initial set of observations $\{\mathbf{o}^1_{t_0}, \ldots, \mathbf{o}^N_{t_0}\}$ and a set of goals $\mathbf{G}$, compute a set of functions $\mu_n(\mathbf{o}^n_t)$ for all$n = 1, \ldots, N$, for all$t$ such that applying the actions $\{a^t_1, \ldots, a^t_N\} := \{\mu_1(\mathbf{o}^1_t), \ldots \mu_N(\mathbf{o}^N_t)\}$ results in a sequence of observations each satisfying Eqn. 2.5 and at final time $t = T$ satisfies Eqn. 3.13. In the next section we outline our methodology to easily convert this problem to a Markov game for MARL and outline

our methodology to compute policies such that the constraints in Eqn. 2.5 and Eqn. 3.13 are satisfied.

2.2 LEARNING UNLABELED MULTI-ROBOT PLANNING

2.2.1 Markov Games for Multi-Robot Planning

One can reformulate the unlabeled multi-robot planning problem describe in Problem 2.1.1 as a Markov game [24]. A Markov game for N robots is defined by a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{R}, \gamma\}$. $\mathcal{S}$ describes the full true state of the environment. We observe corresponds to the full state space in the problem setup. Similarly we observe that the action space $\mathcal{A}$ is the same as defined before in Problem 2.1.1. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is equivalent to the dynamics distribution $p(\mathbf{o}_{t+1}|\mathbf{o}_t, a_t)$ describe in Section 2.1. In the Markov game, the environment is partially observed. At every timestep, each robot can only observe some small part of the environment. This is also defined in our problem and for every timestep t we can simply set the observation for robot $\mathcal{O}_n = \mathbf{o}_t^n$, and $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$. γ is a discount factor and can be set close to one. In the Markov game, each robot n obtains a reward function as a function of robots state and its action. We propose formulating a reward structure that satisfies the constraints in Eqn. 2.5 and Eqn. 3.13.

$$r(t) = \begin{cases} \alpha & \text{if } \phi(t)^\top \phi(t) = \mathbf{I}_N \\ -\beta, & \text{if any collisions} \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

where α and β are some positive constants. It is important to note that this reward structure is global and is given to all robots $\{r(t) = r_1(t), =, r_n(t)\}$. By using a global reward we remove the need for any carefully designed heuristic function. In the Markov game, the solution for each robot n is a policy $\pi_n(a^n|\mathbf{o}^n)$ that maximizes the discounted expected reward $R_n = \sum_{t=0}^{T} \gamma^t r_n(t)$. Once again, we draw parallels between the Markov game and **Problem 1** and set $\mu_n = \pi_n$. Thus, we can conclude the solution of the Markov game for Multi-Robot Planning is the solution for the Unlabeled multi-robot planning considered in Problem 2.1.1.

2.2.2 Learning Policies for Continuous Actions

Consider a single robot setting. The MDP assosciated with a single robot is given as $\mathcal{M}_t(\mathcal{O}, \mathcal{A}, \mathcal{T}_1, r, \gamma)$ ($\mathcal{T}_1$ is the transition function associated with just the robot under consideration). The goal of any RL algorithm is to find a stochastic policy $\pi(\mathbf{o}_t|a_t;\theta)$ (where θ are the parameters of the policy) that maximizes the expected sum of rewards :

$$\max_{\theta} \mathbb{E}_{\pi(a_t|\mathbf{o}_t;\theta)}[\sum_{t} r_t] \tag{2.9}$$

Policy gradient methods look to maximize the reward by estimating the gradient and using it in a stochastic gradient ascent algorithm. A general form for the policy gradient can be given as:

$$\hat{g} = \mathbb{E}_t[\nabla_\theta \log \pi_\theta(a_t|\mathbf{o}_t) Q_t^\pi(\mathbf{o}_t, a_t)] \tag{2.10}$$

where $Q^\pi(o_t, a_t)$ represents the action value function (estimate of how good it is to take an action in a state)

$$Q^\pi(\mathbf{o}_t, a_t) := \mathbb{E}_{\mathbf{o}_{t+1:\infty}, a_{t+1:\infty}}\Big[\sum_{l=0}^{\infty} r_{t+l}\Big] \tag{2.11}$$

Bellman equations also give us the recursive rule for updating the action value function given as :

$$Q^{\pi}(\mathbf{o}_t, a_t) = \mathbb{E}_{\mathbf{o}_{t+1:\infty}, a_{t:\infty}}\Big[r(\mathbf{o}_t, a_t) + \gamma \mathbb{E}_{\mathbf{o}_{t+1:\infty}, a_{t+1:\infty}}(Q^{\pi}(\mathbf{o}_{t+1}, a_{t+1}))\Big] \tag{2.12}$$

where $r(\mathbf{o}_t, a_t)$ is the reward for executing action a_t in $\mathbf{o}_t$. The gradient $\hat{g}$ is estimated by differentiating the objective wrt θ:

$$L^{PG}(\theta) = \mathbb{E}[\log(\pi_\theta(a_t|\mathbf{o}_t)Q^{\pi}(\mathbf{o}_t, a_t)] \tag{2.13}$$

In order to extend this to continuous actions and continuous deterministic policies, [25] propose the the Deep Deterministic Policy Gradient (DDPG) algorithm for continuous actions and deterministic policies. The algorithm maintains an actor function (parameterized by θ^{π}) that estimates the deterministic continuous policy π. In addition, it also maintains a critic function (parameterized by θ^{Q}) that estimates the action value function. The critic function is updated by using the Bellman loss as in Q-learning [26] (Eqn. 2.12) and the actor function is updated by computing the following policy gradient :

$$\hat{g} = \mathbb{E}_t[\nabla_\theta \log \pi_\theta(a_t|\mathbf{o}_t)\nabla_a Q_t^{\pi}(\mathbf{o}_t, a_t)] \tag{2.14}$$

The DDPG algorithm is an off-policy algorithm and samples trajectories from a replay buffer of experiences stored in a replay buffer. Similar to DQN [10] it also uses a target network to stabilize training.

A natural question to ask at this point is, why not treat every robot in the space as an entity operating independently and learn this DDPG algorithm for each robot and in fact this exact idea has been proposed in *Independent Q-Learning* [27]. However, there are two major drawbacks to this approach. When operating in high dimensional continuous spaces

with sparse rewards, the lack of information sharing between the robots makes it difficult to learn any co-ordination between robots. Further, as each robot's policy changes during training, the environment becomes non-stationary from the perspective of any individual robot (in a way that cannot be explained by changes in the robots's own policy). This is the non-stationarity problem in multi-agent learning [26].

2.2.3 Learning Continuous Policies for Multiple Robots

To overcome the aforementioned drawbacks in treating each robot as an independent entity, a small modification to the critic function (action-value) during training time is needed. During training, the critic function for robot n uses some extra information h from all other robots. This has been proposed in [22, 23]. The modified action value function for robot n can then be represented as $Q_n((h_1(t),\dots,h_N(t)),(a_1(t),\dots,a_N(t)))$. The most naive method is to simply set $h_n(t) = \mathbf{o}_n(t)$.

Let policy for robot n parameterized by θ_n be $\pi_n^{\theta_n}$. For brevity sake, let $\{h_1(t),\dots,h_N(t)\} = \mathbf{H}$, $\{a_1(t),\dots,a_N(t)\} = \mathbf{A}$ and $\Pi = \{\pi_1^{\theta_1},\dots,\pi_N^{\theta_N}\}$ Thus in multi-robot case, the gradient of the actor function for robot n is given as

$$\hat{g}_n = \mathbb{E}_t[\nabla_{\theta_n} \log \pi_n^{\theta_n}(a_n(t)|\mathbf{o}_n(t)) \nabla_{a_n} \hat{Q}_n^{\Pi}(\mathbf{H},\mathbf{A})] \tag{2.15}$$

where $\hat{Q}_n^{\Pi}(\mathbf{H},\mathbf{A})$ is the centralized critic function for robot n that takes in input all robot observations and all robot actions and outputs a q value for robot n. The robot n then takes a gradient of this q value with respect to to the action a_n executed by robot n and this gradient along with the policy gradient of robot n's policy is used to update the actor function for robot n. It is important to note that the extra information from other robots actions is only used during training to update the critic function. This gives rise

to centralized training but decentralized policies during inference. Thus, we now have a policy gradient algorithm that attempts to learn a policy for each robot such that Problem 2.1.1 captured in Eqn. 2.9 is maximized.

2.2.4 Model Based Backup Policies for Safety

When using deep RL, due to the stochastic nature of the solution one often loses any guarantees of safety. Thus, when attempting to maximize the reward in Eqn. 2.9, we have no guarantee that actions generated by our actor network are collision free (satisfy constraint in Eqn. 2.5). In real world applications of robotics this could be simply infeasible. Instead, we propose use of a simple analytical backup policy that ensures collision free trajectories.

We use the Velocity Obstacle concept introduced in [28]. While there exist more sophisticated algorithms for collision avoidance such as ORCA [29] and NH-ORCA [30], we opt for VO due to its simplicity. Consider a robot n, operating in the plane with its reference point at $\mathbf{p}_n$ and let another planar obstacle b (another robot or a static obstacle), be at $\mathbf{p}_b$, moving at velocity $v_b(t)$. The velocity obstacle $VO_b^n(v_b(t))$ of obstacle b to robot n is the set consisting of all those velocities $v_n(t)$ for robot n that will result in a collision at some moment in time with obstacle b. The velocity obstacle (VO) is defined as:

$$VO_b^n(v_b(t)) = \{v_n(t) | \lambda(\mathbf{p}_n, v_n(t) - v_b(t)) \cap b \bigoplus -n \neq \emptyset\}$$

where $\bigoplus$ gives the Minkowski sum between object n and object b, $-n$ denotes reflection of object n reflected in its reference point $\mathbf{p}_n$, and $\lambda(\mathbf{p}_n, v_n(t) - v_b(t))$ represents a ray starting at $\mathbf{p}_n$ and heading in the direction of the relative velocity of robot n and b given by $v_n(t) - v_b(t)$. [28] show that the VO partitions the absolute velocities of robot n into

avoiding and colliding velocities. This implies that if $v_n(t) \in VO_b^n(v_b(t))$, then robot n and obstacle b will collide at some point in time. If $v_n(t)$ is chosen such that it is outside the VO of b, both objects will never collide and if v_n is on the boundary, then it will brush obstacle b at some point in time.This concept is illustrated in Fig. 2.2.

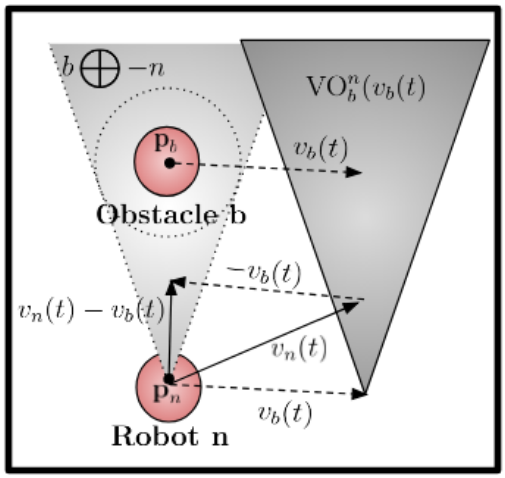


Figure 2.2: Velocity Obstacle Velocity obstacle $VO_b^n(v_b(t))$ of obstacle b to robot n. When there exist multiple obstacles, the VO is defined as the union of all velocity obstacles.

Each robots actor network outputs a linear force and a torque, i.e $a_n(t) = \{\mathbf{F}_n(t), \tau_n(t)\}$. The dynamics of the robot then evolve according to :

$$[v_n(t+1)] := \left[\frac{\mathbf{F}_n(t) + z}{\kappa}\right](\Delta) \tag{2.16}$$

$$[x_n(t+1), y_n(t+1)] := v_n(t+1)\Delta + [x_n(t), y_n(t)] \tag{2.17}$$

where z is some normally distributed noise $z \sim \mathcal{N}(0, \sigma^2)$, κ is mass of the robot and is some fixed constant and Δ is the fixed time interval. Similarly, the rotational acceleration is derived from the torque and integrated over twice to update the orientation. For simplicity, say that the observation is set to just the position and the velocity in the 2D plane, i.e

$\mathbf{o}_t = [\mathbf{p}_t, v_n(t)]$. From Eqn. 2.16, we derive the stationary dynamics distribution with conditional density as $p(\mathbf{o}_{t+1}|\mathbf{o}_t, a_t) \sim \mathcal{N}(\frac{\Delta^2\mathbf{F}(t)+x_t}{\kappa}, \frac{\sigma^2\Delta^4}{\kappa^2})$

A fundamental assumption for the existence of the gradient in Eqn. 2.14 (and by extension, Eqn. 2.15) is that the conditional probability distribution $p(\mathbf{o}_{t+1} = \xi|\mathbf{o}_t, a_t)$ be continuous wrt $\mathbf{o}_t$, $\mathbf{a}_t$ for allξ . We observe that in the case when the state evolves according to Eqn. 2.16, the probability distribution is simply a gaussian distribution and is continuous. In order to incorporate the VO as a backup policy, we need to prove that the new transition function is still continuous. From Eqn. 2.16, we have the velocity of the robot. Further, from Eqn. 2.17 $x_{t+1} = \Delta v_n(t) + x_t$. Since x_t is a fixed, it suffices to find the continuity of $p(v_n(t))$ to conclude about the continuity of $p(\mathbf{o}_{t+1}|\mathbf{o}_t, a_t)$. Consider an obstacle B inside sensing range of robot n. To ensure safety, at every timestep, we compute the VO and check if the velocity $v_n(t) \in VO^n_B(v_b(t))$. In case, the velocity computed by the actor network falls inside the VO, we project the velocity back to the safe set VO' (VO' is the complement of the VO). The easiest projection can be given as :

$$P^{min}_{VO'}(v_n(t)) = \{\min_{\bar{v}} ||v_n(t) - \bar{v}|| : \bar{v} \in VO'\} \tag{2.18}$$

Thus, the safe velocity for robot n is then given as:

$$v^{safe}_n(t) = v_n(t)\mathbb{1}_{VO'}(v_n(t)) + P^{min}_{VO'}(v_n(t))(1 - \mathbb{1}_{VO'}(v_n(t)) \tag{2.19}$$

where $\mathbb{1}_a(b)$ is the indicator function and takes the value 1 if $b \in a$ and 0 otherwise. However, an issue with such a projection is that this gives us a discontinuous distribution for $p(v^{safe}_n(t))$ (probability) because now, there exists a set of values that $v^{safe}_n(t)$ never takes. Thus, this fails the assumption of smooth transition functions necessary to compute deterministic policy gradients. Additionally, in most real world systems, it is infeasible to make large changes to the velocity instantaneously.

To overcome this, we propose an alternate projection that ensures a smooth distribution of $v_n^{safe}(t)$. We note that at any given time t, the RVO set, i.e the set of infeasible velocities is always a continuous set. By exploiting this property, we propose the following alternative projection:

$$P^{sig}_{VO'}(v_n(t)) = \frac{v_i - v_k}{1 + e^{-c(v_n(t) - v_j)}} + v_k \tag{2.20}$$

where v_i, v_j, v_k are the first, middle and last elements of the VO respectively and c is a hyperparameter that depends on the how quickly the robots can change their velocities. This is a shifted sigmoid projection and is visualized in Fig. 2.3. Using the sigmoid

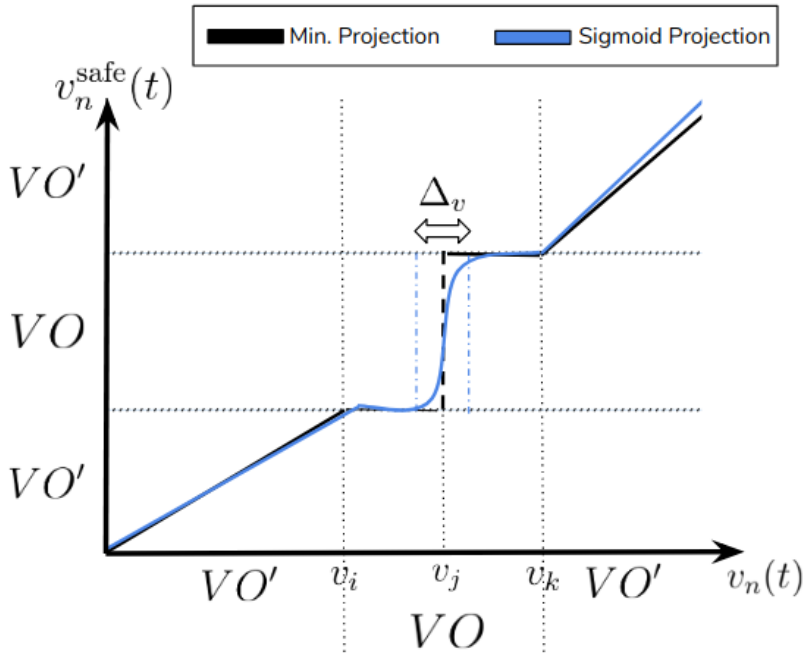


Figure 2.3: Min Projection vs Sigmoid Projection of velocity When using the minimum projection given in Eq 2.18 the safe velocity has a discontinuity. However, if we assume that the event $v_n(t) = v_j \pm \Delta_v$ occurs with very low probability, we get $p(v_n^{safe}(t))$ a continuous function.

projection, the safe velocity for robot n is given as :

$$v_n^{safe}(t) = v_n(t)\mathbb{1}_{VO'}(v_n(t)) + P^{sig}_{VO'}(v_n(t))(1 - \mathbb{1}_{VO'}(v_n(t)) \tag{2.21}$$

and from this we conclude that $p(\mathbf{o}_{t+1}|\mathbf{o}_t, a_t)$ is a continuous probability distribution thus enabling us to take the gradients specified in Eqn. 2.15. We put all these parts into our system for learning unlabeled multi-robot planning with motion constraints and present the full algorithm in Algorithm 1 and in the rest of the paper we reference it as **MARL+RVO**

Algorithm 1 Learning Safe Unlabeled Multi-Robot Motion Planning (MARL+RVO)

Require: Initial random policy network and critic networks for all robots Π Replay buffer D,

for episode = 1 to C (C >> 1) **do**
  construct every robot's initial state $\mathbf{o}_t^n$ (Eq 2.3).
  for t= 1 to max episode length **do**
    for each robot, compute $a_t^n = \pi_n(\mathbf{o}_t^n)$
    for each robot, guarantee *safe* a_t^n (Eqn. 2.21)
    for each robot, compute $\mathbf{o}_{t+1}^n$ and reward $r(t)$.
    Store $\mathbf{o}_{t+1}^n, a_t^n, \mathbf{o}_t^n, r(t)$ in D
    for robot = 1 to N **do**
      Sample minibatch of samples from D
      Compute bellman error using Eqn. 2.12
      Update critic network using bellman error.
      Compute policy gradient $\hat{g}_n$ from Eqn. 2.15
      Update actor network using SGD with $\hat{g}_n$
    end for
  end for
end for

2.3 EXPERIMENTAL RESULTS

The efficacy of our algorithm is tested in simulated robotics experiments. We experiment by changing the number of robots, number of obstacles present in the environment and the robot dynamics. In order to choose a meaningful reward function that ensures all goals are covered we first compute for each goal the distance to its nearest robot. Then among this

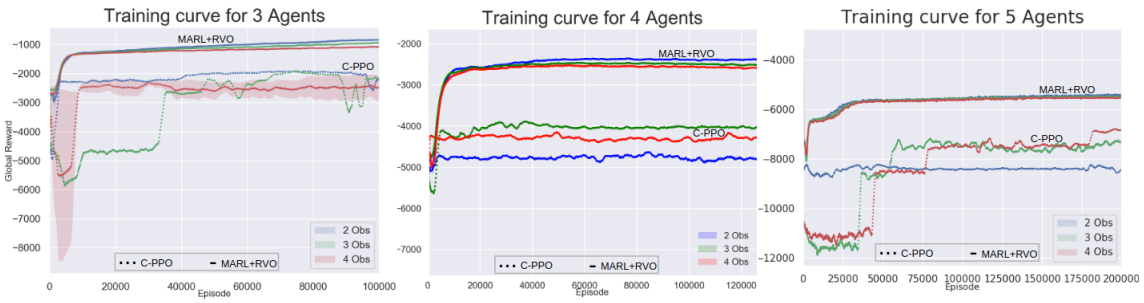


Figure 2.4: Training curves for holonomic robots (with 2,3 and 4 Obstacles (Obs)). We observe that the proposed MARL+RVO algorithm is able to converge and perform better than a centralized RL (C-PPO) policy. The global reward scale is different for each plot since it is a function of the space the robots operate in. Each curve is produced by running three independent runs of the algorithm. Darker line represents mean and shaded area represents mean $\pm$ standard deviation of mean.

set of distances, we pick the maximum, negate it and add it to the reward. This represents the part of the reward function that forces all robots to cover all goals (denoted by $r_D(t)$). A similar strategy is adopted to ensure that the cosine difference between orientations is minimized (denoted by $r_r(t)$). In order to not overly depend on the projected velocity, we add in a negative penalty every time the projection to the safe set needs to be computed. Thus, we add a negative reward to all robots (denoted by $r_C(t)$). Thus, the overall reward given to each robot at time t is :

$$r(t) = \lambda_D r_D(t) + \lambda_r r_r(t) + \lambda_C r_C(t) \tag{2.22}$$

where λ_D, λ_r and λ_C are coefficients to balance each part of the reward function. This global reward function is the same for every robot operating in the environment. Maximizing this global reward requires a collective effort from all robots.

It is important to note that during, inference time to guarantee safety, we do away with the soft projection introduced in Eqn. 2.20 and instead use the min projection as given in

Eqn. 2.18. This is because during inference we no longer need to take gradients and hence the transition function need not be smooth continuous anymore.

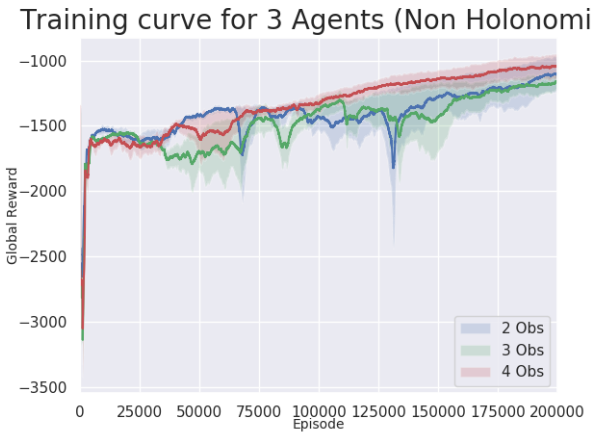


Figure 2.5: Training curves for non holonomic robots (with 2,3 and 4 Obstacles (Obs)) When the robot dynamics are changed, MARL+RVO is still able to converge without making any changes to the loss function or the training parameters.

2.3.1 Experimental Setup Details

For each robot, we setup an actor and critic network. The actor network consists of a two layer fully connected multi-layer perceptron (MLP). The critic network is also based on a similar fully connected MLP. The number of units in the hidden layers are varied depending on the size of the problem being solved (additional units and hidden layers when number of robots or obstacles are increased). For each episode, we set a maximum episode length of 300 steps. To update our networks, we use Adam and the learning rate is varied depending on the experiment under consideration. The discount factor (γ) is set to 0.95 We also make use of a replay buffer to make sure dependencies between samples are modelled. The size of the replay buffer is 10^5 and the size of the minibatch sampled is 1024. The actions from the neural networks represent accelarations for the

robots. The robots operate in a two dimensional space and do not have access to the third

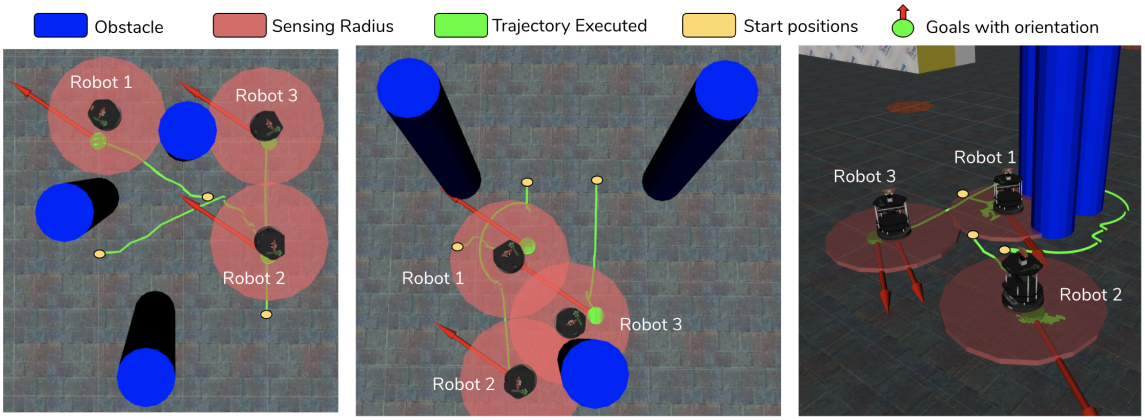


Figure 2.6: Trajectory executed by 3 robots Trajectories executed by robots in three randomly generated episodes after training is complete. In addition to robots reaching their goals in collision free manners, the proposed approach also aligns robots to desired final goal orientations.

dimension. The space under consideration stretches from -1 unit to 1 unit in both the X and Y direction. Each robot is considered homogeneous and has nonzero mass. The radius of the robot is set to 0.05 units. At the start of every episode obstacles, goals and start positions of robots are randomly populated. Radius of the obstacles is 0.12 units and the goal regions have a radius of 0.02 units. Robots are equipped with a sensor that returns perfect information (no noise in sensor measurements) about the pose and velocity of entities within the sensing range which is set at radius of 0.2 units. While the learning algorithm does not have an explicit assignment of goals in the states or in the reward function, when using RVO to avoid collisions, we need to greedily assign goals to agents based on distance to nearest goal and break all ties by randomly choosing goals. Once the RVO subprocess is done running, we again have no notion of goal assignment. In our simulated experiments (below) we set all units to meters.

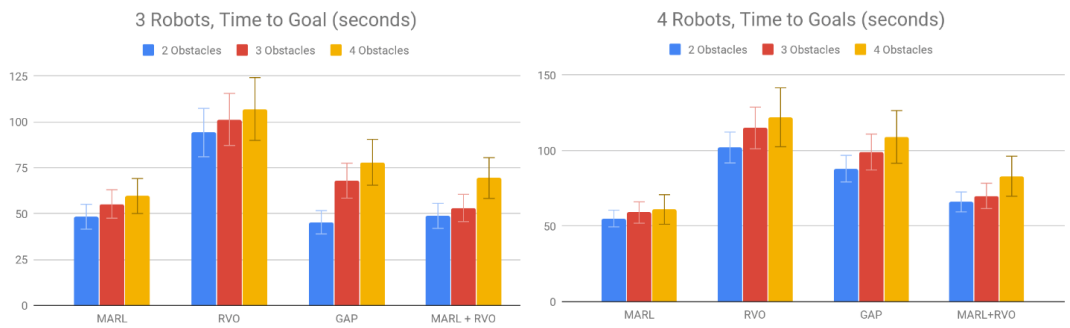


Figure 2.7: Time taken to reach goal (with 2,3 and 4 Obstacles (Obs)) over 500 runs. We compare with Multi-Agent Reinforcement Learning (MARL), Reciprocal Velocity Obstacles (RVO), GAP [21] and our algorithm which combines the RL and safety(MARL+RVO). For the RVO method, we assign each robot its nearest goal (in terms of euclidean distance). GAP uses discrete nodes to search through space and hence its performance is contingent on the discretization of our continuous space. We observe that with MARL gives the best time performance, but this performance is not guaranteed to be collision free. Our method (MARL+RVO) trades-off time performance for guaranteed collision free trajectories.

2.3.2 Simulation Results

We first observe from Fig. 2.4 that the proposed MARL+RVO algorithm is able to converge even when the number of robots and the number of obstacles are increased. One of the key strengths of using a learning based solution for concurrent goal assignment and planning is that the algorithm can be used even when the dynamics of the robot change. When robot dynamics are changed to that of a non holonomic robot, we observe that our algorithm still converges. This can be seen in Fig. 2.5. A simulated experiment setup is shown in Fig 2.6. In Fig. 2.6 (left), a simple instance is shown where all robots must execute mostly straight line trajectories to arrive at goals. In Fig. 2.6 (center), an interesting interaction takes place between Robots 1 and 2. Robot 2 takes a longer path curved path around Robot 1. Lastly, in Fig. 2.6 (right) Robot 3 chooses to take a longer path around the obstacles in order to not cutoff Robot 2's path. These locally sub optimal, globally optimal behaviors

are induced by using a global reward. We also design a centralized RL controller that uses information from all the agents and outputs a distribution for each agent. This controller is trained using PPO [31] and also uses velocity obstacles as a backup policy. We call this method Centralized PPO (C-PPO). We observe that C-PPO is unable to converge to an acceptable goal coverage policy.

In the RL framework, the policy attempts to maximize the reward function in a fixed horizon of time T. Thus, inherently the policy is being optimized for minimum time. To demonstrate this we compare our algorithm with vanilla MADDPG or MARL as described in [22], reciprocal velocity obstacles [32] (RVO), and Goal Assignment and Planning (GAP) as introduced in [21]. The RVO framework improves over VO. However, it is not a full "goal assignment and planning" framework and only generates collision free trajectories once goals have been assigned to robots. To benchmark, we assign goals in a greedy fashion. Each robot is assigned the goal closest to it. GAP utilizes a similar assignment but needs a discretization of the state space and a priority sequence for robots/goals. This prioritization is assigned randomly and the space is discretized into units of 0.1m. Out of these three methods, only RVO, GAP and MARL+RVO are guaranteed to produced collision free trajectories. For a fair comparison in terms of time, we only consider those runs from MARL where no collision occurred. Our results are shown in Fig. 2.7. It can be seen that MARL and MARL+RVO is faster or almost comparable to GAP and RVO without needing any of the requirements of (assigning goals/discretized state space/priority sequence) GAP and RVO. Vanilla MARL is faster than MARL+RVO but isn't guaranteed to generate safe trajectories as seen from Table 2.1.

	3 Robots	4 Robots	5 Robots
MARL	84	192	354
MARL+RVO	**0**	**0**	**0**

Table 2.1: Number of collisions for 3 robots in presence of 3 obstacles over 500 runs.

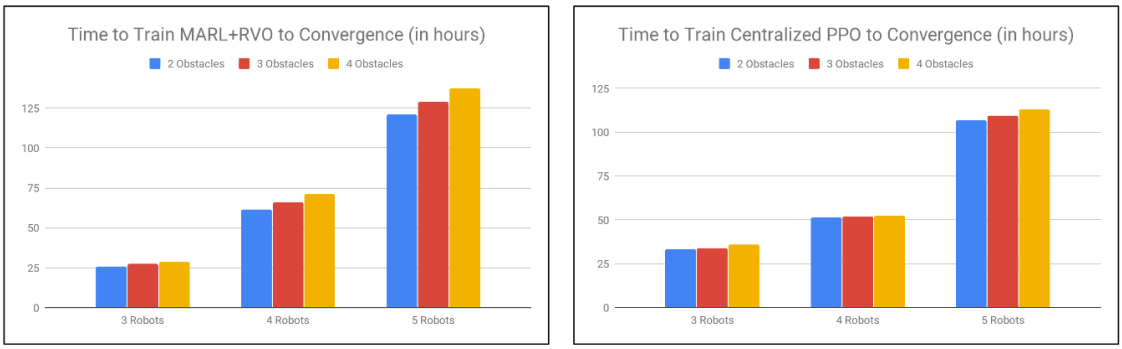


Figure 2.8: Training time to convergence. Training time for different configurations of robots and agents when trained on a NVIDIA DGX-1 (Tesla V100, 32GB $\times$ 8)

2.4 DISCUSSION

In this paper we propose to solve the concurrent goal assignment and planning problem using MARL instead. Traditional approaches to solve this problem utilize a carefully designed heuristic function which produces guaranteed safe trajectories but breaks down if any of the assumptions are not satisfied. These assumptions restrict the class of problems that can be solved by traditional algorithms. By utilizing RL, we remove any assumptions on the robot dynamics or assumptions on the environment and instead use a global reward function that forces robots to collaborate with each other in order to maximize the reward. To overcome the lack of any safety guarantees, we propose using a model based policy in conjunction with the RL policy thus ensuring safe collision free trajectories. We demonstrate the effectiveness of our algorithm on simulations with varying number of obstacles, varying number of robots and varying robot dynamics and show that our proposed algorithm works faster and more robustly than traditional algorithms.

2.4.1 Caveats

While this work attempts to learn an approximate solution for the unlabeled multi-robot problem, it has a few caveats. One of the biggest drawbacks of our work is that there is a significant engineering effort required in scaling up the number of robots. When the number of robots is increased, there are two major challenges that hamper MARL. The first is that the input space of the critic function grows as the number of robots increase. This increase in dimensionality necessitates longer training times for the critic. It might be possible to instead propose a local critic function that only takes in information from nearby robots. This might be possible by thinking of the robots as nodes on a graph and leveraging advances in graph neural networks [33], instead of using a fully connected network. The second problem is concerned with the need for more exploration as the number of robots increase. We observe from Fig 2.8 that the time required to train the algorithm grows almost exponentially as the number of robots are increased. While there exist massively parallel methods [34, 35] and software libraries [36] to scale up for reinforcement learning, scaling up the number of robots still poses a significant computing challenge. Methods attempting to learn hierarchical policies for agents such as those in [37] might instead prove to be a suitable alternative. Lastly, our choice of VO for collision avoidance while rooted in its simplicity suffers from drawbacks many of which have been improved over by methods presented in [29], [38].

2.4.2 Guiding Ideas

One of the key insights that can be derived from this work is that thee underlying symmetry of the robots and the locality of the problem strucuture; we hypothesize that during training if one can learn to recognize the underlying symmetry existing in the

problem, then it might be possible to scale the solutions presented in this work to a larger solution set. In the next chapters, we hypothesize using graph convolutions to parametrize individual robot policies and also devise new methods to scale training to large number of robots.

3 GRAPH POLICY GRADIENTS FOR LARGE SCALE ROBOT CONTROL

Chapter 2 introduced the idea of using multi-agent reinforcement learning as a tool for computing solutions for the distributed decentralized control of multiple robots for concurrent goal assignment . However, a key caveat of these methods was that as the number of robots increases, the dimensionality of the input space and the control space both increase making it much harder to learn meaningful policies.

In this chapter we look to tackle the problem of learning individual control policies by exploiting the underlying graph structure among the robots. We start with the hypothesis that the difficulty of learning scalable policies for multiple robots can be attributed to two key issues: dimensionality and partial information. Consider an environment with $\mathbf{N}$ robots (In this chapter and henceforth we shall use bold font when talking about a collection of items, vectors and matrices). Each robot receives partial observations of the environment. In order for a robot to learn a meaningful control policy, it must interact with some subset of all agents, $\mathbf{n} \subset \mathbf{N}$. Finding the right subset of neighbors to learn from is in itself a challenging problem. Further, in order to ensure that the method scales, as the number of robots increase, one needs to ensure that the cardinality of the subset of neighbors $|\mathbf{n}|$, remains fixed or grows very slowly. To solve these problems for large scale multi-robot control, we draw inspiration from convolutional neural networks (CNNs). CNNs consist of sequentially composed layers where each layer comprises of banks of linear time invariant filters to extract local features along with pooling operations to reduce the dimensionality. Convolutions regularize the linear transform to exploit the underlying structure of the

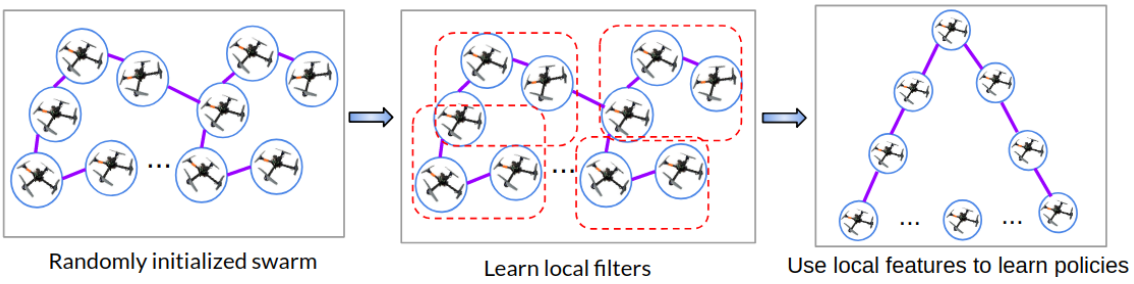


Figure 3.1: Graph Policy Gradients. Robots are randomly initialized and, based on some user set thresholds, a graph is defined. Information from K-hop neighbors is aggregated at each node by learning local filters. These local features are then used to learn policies to produce desired behavior.

data and hence constrain the search space for the linear transform that minimizes the cost function. However, CNNs cannot be directly applied to irregular data elements that have arbitrary pairwise relationships defined by an underlying graph. To overcome this limitation, there has been the advent of a new architecture called graph neural networks (GNNs) [33, 39, 40]. Similar to CNNs, GNNs consist of sequentially composed layers that regularize the linear transform in each layer to be a graph convolution with a bank of graph filters and the weights of the filter are learned by minimizing some cost function.

When controlling a large swarm of robots operating in a Euclidean space $\mathbb{R}^n$, the underlying graph can be defined as $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ where $\mathbf{V}$ is the set of nodes and $\mathbf{E}$ is the set of edges. For example, we define each robot to be a node and add an edge between robots if they are are less than ϵ distance apart. This graph acts as a support for the data vector $\mathbf{x} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top$ where $\mathbf{x}_n$ is the state representation of robot n. Now, our GNN exploits this graph $\mathcal{G}$ and at each node aggregates information from its neighbors (See Section 3.1.2). This information is propagated forward to the next layer through a non linear transformation. The output of the final layer is given by $\Pi = [\pi_1, \ldots, \pi_N]$, where $\pi_1, \ldots, \pi_N$ are independent policies for the robots. Similar to standard reinforcement learning, we execute these policies in the environment, collect a centralized reward and

use policy gradients [26] to update the weights of policy network. We call this algorithm Graph Policy Gradients (GPG) (See Fig. 5.1).

One possible concern with our proposed algorithm is that, when working with on-policy methods for a large number of robots, it is highly likely that, due to exploration/entropy the graph might change during training. Such a setting can result in an explosion in the number of possible graphs that one needs to learn over as the number of robots increases. We show that in light of this, our choice of GNNs for policy parametrization is well motivated because it can be shown that graph convolutions are permutation equivariant, i.e if one were to reorder the node ordering of the graph and the corresponding graph signal, then the output of the graph convolution does not change [41]. This is important because it reduces the dimensionality of the problem and helps training converge faster when training with many robots.

To demonstrate the efficacy of our proposed algorithm, we perform experiments on simulated formation flying with robots. Designing controllers and trajectories for formation flying is an important problem in multi-robot literature [42, 43] and is in fact a good test bed for other multi-robot control problems [44]. In our experiments, it is shown that GPG is able to converge as the number of robots are increased in comparison to state of the art on-policy reinforcement learning algorithms that employ fully connected networks to parametrize the policy. We also show that our graph filters are able to learn valid local features by training them on a small number of robots and transferring the behavior to a very large number of robots. For example, we train a GNN for three robots to maintain formation, avoid collisions and follow their trajectories. Then, we initialize a swarm of many robots and use this same graph filter over the entire graph to generate desired behavior. We show that the desired formation flying behavior is achieved without updating the weights for the larger swarm thus achieving zero-shot transfer. Lastly, the ability of GPG to adapt to more complex dynamics and control is demonstrated.

3.1 METHODOLOGY

3.1.1 Preliminaries

We pose learning the controller for a large number of robots as a policy learning problem in a collaborative Markov team [24]. The team is composed of $\mathbf{N}$ robots generically indexed by n which at any given point in time t occupy a position $\mathbf{x}_{nt} \in \mathcal{X}$ in configuration space and must choose an action $\mathbf{a}_{nt} \in \mathcal{A}$ in action space. The team and environment are assumed to be Markovian. Thus, if one were to collect the robot configurations in the vector $\mathbf{x}_t := [\mathbf{x}_{1t}, \dots, \mathbf{x}_{Nt}]^\top$ and actions in the vector $\mathbf{a}_t := [\mathbf{a}_{1t}, \dots, \mathbf{a}_{Nt}]$ then, the evolution of the system is completely determined by the conditional transition probability $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t)$. The transition dynamics are also assumed to be the same for all agents, so that if we swap two of them in configuration and action space we expect to see the same statistical evolution. This is a natural consequence when, the robotic swarm is assumed to be composed of homogeneous robots. In this work, there exists no external communication between robots. Instead, we define a graph $\mathcal{G}$ that encodes information about each robots neighbors. Robot n can directly communicate only with its one hop neighbors. Communication with neighbors that are further away is possible only indirectly. Each robot has access to its own states $\mathbf{x}_{nt}$ and $\mathcal{G}$. In this paper, we do not design which nodes each robot should talk to or what each robot must communicate. Each robot must learn a policy (probability distribution) from which the robot samples actions; $\mathbf{a}_{nt} := \pi_n(\mathbf{a}_{nt}|\mathbf{x}_{nt}, \mathcal{G})$. Let $\Pi = [\pi_1, \dots, \pi_n]$. As robots operate in the environment, they collect a *centralized* global

reward r_t. Collapsing everything, we are interested in computing $\mathbf{a}_t := \Pi(\mathbf{a}_t|\mathbf{x}_t, \mathcal{G})$ such that the expected sum of rewards over some time horizon T is maximized:

$$\sum_{n=1}^{N} \max_{\theta} \mathbb{E}_{\Pi}\left[\sum_{t}^{T} r_t\right] \tag{3.1}$$

where θ are the parameters of Π. At a high level, the global reward encodes desired behavior for the swarm. As the number of robots increase, the dimensionality of $\mathbf{a}_t$ and $\mathbf{x}_t$ increases too making the problem of learning $\Pi(\mathbf{a}_t|\mathbf{x}_t)$ non-trivial. Our proposition here to instead learn $\Pi(\mathbf{a}|\mathbf{x}, \mathcal{G})$ only compounds the difficulty of the problem as the size of the graph grows exponentially as the number of robots increase (N^2 for N robots). In the next section, we discuss using a graph convolutional network as a parametrization for Π to overcome this problem by extracting local information from the graph structure.

3.1.2 Graph Neural Networks

Consider a graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ described by a set of $\mathbf{N}$ nodes denoted $\mathbf{V}$, and a set of edges denoted $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$. This graph is considered as the support for a data signal $\mathbf{x} = [x_1, \ldots, x_N]^\top$ where the value x_n is assigned to node n. The relation between $\mathbf{x}$ and $\mathcal{G}$ is given by a a matrix $\mathbf{S}$ called the graph shift operator. The elements of $\mathbf{S}$ given as s_{ij} respect the sparsity of the graph, i.e $s_{ij} = 0$, for all $i \neq j$ and $(i, j) \notin \mathbf{E}$. Valid examples for $\mathbf{S}$ are the adjacency matrix, the graph laplacian, and the random walk matrix. In this paper, we consider the normalized graph Laplacian similar to [33]:

$$\mathbf{S} = \mathbf{I}_N - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \tag{3.2}$$

A key property of $\mathbf{S}$ is that it is assumed symmetric, with decomposition $\mathbf{S} = \mathbf{V}\Lambda\mathbf{V}^\top$ where $\mathbf{V}$ is the eigenvector matrix and Λ is the eigenvalue matrix of $\mathbf{S}$. $\mathbf{S}$ defines a map $\mathbf{y} = \mathbf{S}\mathbf{x}$ between graph signals that represents local exchange of information between a node and its one-hop neighbors. More concretely, if the set of neighbors of node n is given by $\mathcal{B}_n$ then :

$$y_n = [\mathbf{S}\mathbf{x}]_n = \sum_{j=n, j\in\mathcal{B}_n} s_{nj} x_n \tag{3.3}$$

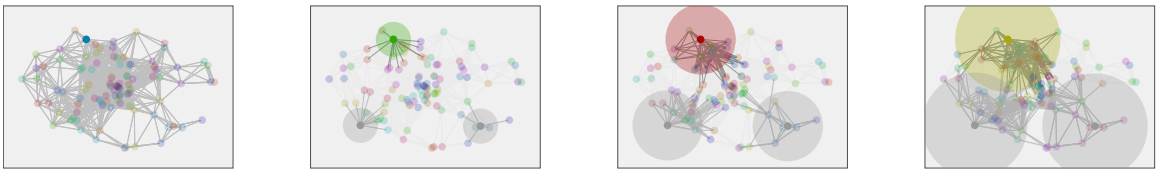
Eq. 3.3 performs a simple aggregation of data at node n from its neighbors that are one-hop away. The aggregation of data at all nodes in the graph is denoted $\mathbf{y} = [y_1, \ldots, y_N]$. By repeating this operation, one can access information from nodes located further away. For example, $\mathbf{y}^k = \mathbf{S}^k\mathbf{x} = \mathbf{S}(\mathbf{S}^{k-1}\mathbf{x})$ aggregates information from its k-hop neighbors (see Fig 3.2). Now one can define the spectral K-localized graph convolution as :

$$\mathbf{z} = \sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{x} = \mathbf{H}(\mathbf{S})\mathbf{x} \tag{3.4}$$

where $\mathbf{H}(\mathbf{S}) = \sum_{k=0}^{\infty} h_k \mathbf{S}^k$ is a linear shift invariant graph filter [45] with coefficients h_k. Similar to CNNs the output of the GNN is fed into a pointwise non-linear function. Thus, the final form of the graph convolution is given as:

$$\mathbf{z} = \sigma(\mathbf{H}(\mathbf{S})\mathbf{x}) \tag{3.5}$$

where σ is a pointwise non-linearity. A visualization of a GNN can be seen in Fig. 3.2.



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

Figure 3.2: **Graph Convolutional Networks**. GNNs aggregate information between nodes and their neighbors. For each k-hop neighborhood (illustrated by the increasing disks), record $\mathbf{y}_{kn}$ (Eq. 3.3) to build $\mathbf{z}$ which exhibits a regular structure (Eq. 6.6). **a)** The value at each node when initialized and at the **b)** one-hop neighborhood. **c)** two-hop neighborhood. **d)** three-hop neighborhood.

3.1.3 Permutation Equivariance of Graph Convolutional Networks

To control n robots, we propose defining a graph where each robot is a node and robots within ϵ distance of each other are connected by an edge. The robots are all initialized with random policies and by exploring different actions, they learn what policies best optimize the global reward. Such exploration can change the ordering of the configuration $\mathbf{x}$. This can lead to an explosion in the possible number of graphs that our policies have to learn over. However, [41] proves a key property for graph convolutional filters.

Given a set of permutation matrices :

$$\mathcal{P} = \{\mathbf{P} \in \{0,1\}^{N \times N} \ : \mathbf{P}\mathbf{1} = \mathbf{1}, \mathbf{P}^\top \mathbf{1} = \mathbf{1}\} \tag{3.6}$$

where the operation $\mathbf{Px}$ permutes the elements of the vector $\mathbf{x}$ then, it can be shown that :

Theorem 1. *Let graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ be defined with a graph shift operator $\mathbf{S}$. Further, define $\hat{\mathcal{G}}$ to be the permuted graph with $\hat{\mathbf{S}} = \mathbf{P}^\top \mathbf{S}\mathbf{P}$ for $\mathbf{P} \in \mathcal{P}$ and any $\mathbf{x} \in \mathbb{R}^N$ it holds that :*

$$\mathbf{H}(\hat{\mathbf{S}})\mathbf{P}^\top \mathbf{x} = \mathbf{P}^\top \mathbf{H}(\mathbf{S})\mathbf{x} \tag{3.7}$$

The proof for Theorem 1 was first given in [41]. We reiterate it here due to its importance to this body of work.

Proof for Theorem 1:

Given that $\mathbf{P}$ is a permutation matrix. This implies $\mathbf{P}$ is also an orthogonal matrix. This implies $\mathbf{P}^\top\mathbf{P} = \mathbf{P}\mathbf{P}^\top = \mathbf{I}$. Thus,

$$\hat{\mathbf{S}}^k = \mathbf{P}^\top\mathbf{S}^k\mathbf{P} \tag{3.8}$$

Then,

$$\mathbf{H}(\hat{\mathbf{S}}) = \sum_{k=0}^{\infty} h_k\hat{\mathbf{S}}^k = \sum_{k=0} h_k(\mathbf{P}^\top\mathbf{S}^k\mathbf{P}) = \mathbf{P}^\top\mathbf{H}(\mathbf{S})\mathbf{P} \tag{3.9}$$

Finally, using $\mathbf{P}\mathbf{P}^\top = \mathbf{I}$

$$\mathbf{H}(\hat{\mathbf{S}})\mathbf{P}^\top\mathbf{x} = \mathbf{P}^\top\mathbf{H}(\mathbf{S})\mathbf{P}\mathbf{P}^\top\mathbf{x} = \mathbf{P}^\top\mathbf{H}(\mathbf{s})\mathbf{x} \tag{3.10}$$

Hence, proved.

A consequence of Theorem 1 is that the output of the graph convolution does not change under reordering of the graph nodes as long as the topology of the graph stays the same. Intuitively, if the graph exhibits several nodes that have the same graph neighborhoods, then the graph convolution filter can be translated to every other node with the same neighborhood. When learning control for a large number of robots, this property helps in reducing dimensionality of the problem.

3.1.4 Formation Flying

We choose formation flying as a test bed for controlling a large number of robots. In this work, the robots are optimized to produce desired behavior in terms of trajectory following. This behavior can easily be modified from formation flying to other multi-robot

objectives such as flocking [46], information gathering [47], collaborative mapping [48] and multi-robot coverage [49].

Consider a two dimensional Euclidean space $\mathbb{R}^2$ with $\mathbf{N}$ homogeneous point mass robots indexed by n. In later experiments, we use the robot models defined in the AirSim simulator [50] for our experiments. Each robot has a desired goal position $\mathbf{g}_n$ in the euclidean space. Collectively for all robots, all goal locations are denoted $\mathbf{g} = [\mathbf{g}_1, \ldots, \mathbf{g}_N]$. At time t, the robot's position in the plane is given by $\mathbf{p}_{nt}$. The state representation for robot n that is used by our learning architecture consists only of its own relative position to the goal, i.e $\mathbf{x}_{nt} := \mathbf{p}_{nt} - \mathbf{g}_n$. We choose relative positions to goals as a representation of the robots, in order to maintain permutation invariance. It is important to note here that robot n cannot arbitrarily communicate with any other robot in the swarm. Instead we define a graph $\mathcal{G}$ that encodes pairwise relationships between robots. Robot n can directly communicate only with its one hop neighbors. Communication with neighbors that are further away is possible only indirectly. For each robot, given an action $\mathbf{a}_{nt} \in \mathcal{A}$, the state of the robot evolves according to some stationary dynamics distribution with conditional density $p(\mathbf{x}_{nt+1}|\mathbf{x}_{nt}, \mathbf{a}_{nt})$. In this work, we work with continuous control only since it poses a much harder problem and is also more realistic when working with robots. Two conditions are encoded for robots to maintain formation. These are collision avoidance, and waypoint reaching for robots. The necessary and sufficient condition to ensure collision avoidance between robots is given as :

$$E_c(\mathbf{p}_{it}, \mathbf{p}_{jt}) > \delta, \text{for all} i \neq j \in \{1, \ldots N\}, \text{for all} t \tag{3.11}$$

where E_c is the euclidean distance and δ is a user-defined minimum distance between robots. Let us also define an assignment matrix $\phi(t) \in \mathbb{R}^{N \times N}$ as :

$$\phi_{ij}(t) = \begin{cases} 1, & \text{if for all} i = j, E_c(\mathbf{p}_{it}, \mathbf{g}_j) \leqslant \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{3.12}$$

where ϵ is some threshold region of acceptance. The necessary and sufficient condition for all robots to be cover their assigned goals at some time $t = T$ is then:

$$\phi(T)^\top \phi(T) = \mathbf{I}_N \tag{3.13}$$

where $\mathbf{I}$ is the identity matrix. With these definitions in place, we now define the problem statement considered in this paper:

Problem 1. *Given an initial set of robot configurations* $\mathbf{x}_0$*, a graph* $\mathcal{G}$ *defining relationships between robots and some goals* $\mathbf{g}$*, compute a set of policies* $\Pi = [\pi_1, \ldots, \pi_n]$ *such that executing actions* $\{\mathbf{a}_{1t}, \ldots, \mathbf{a}_{Nt}\} = \{\pi_1(\mathbf{a}_{1t}|\mathbf{x}_{1t}, \mathcal{G}), \ldots, \pi_N(\mathbf{a}_{Nt}|\mathbf{x}_{Nt}, \mathcal{G})\}$ *results in a sequence of states that satisfy Eq.3.11 and at time* $t = T$*, satisfy the assignment constraint in Eq.3.13.*

3.1.5 Graph Policy Gradients

We proposing solving the statement in Problem 1 by exploiting the underlying graph structure. Given an initial swarm of N robots, a graph $\mathcal{G}$ can be defined by setting each robot to be a node. Next, an edge is added between nodes, if:

$$E_c(\mathbf{p}_i, \mathbf{p}_j) \leqslant \lambda, \text{for all} i \neq j \in \{1, \ldots N\} \tag{3.14}$$

where λ is some user defined threshold to connect two robots. It is assumed that at time t, the position of robot n (given as $\mathbf{p}_{nt}$) and as a consequence the relative position of robot n (given as $\mathbf{x}_{nt}$) to its own goal is known precisely. The relative positions of all the robots at time t are collected into the vector $\mathbf{x}_t = [\mathbf{x}_{1t}, \ldots, \mathbf{x}_{Nt}]$. The graph $\mathcal{G}$ defined earlier, acts as the support for $\mathbf{x}_t$. We do not evolve the graph over time since interchanging homogeneous nodes results in an equivalent graph convolution as discussed in Section 3.1.3. To ensure that the topology stays constant, the number of neighbors for each node is kept fixed, i.e robot n is connected to its say three or four nearest neighbors. We also experiment with a time varying graph, in our simulations and it is observed that for a small number of robots, policies using computed time varying graphs converge slower than fixed graphs (see Fig. 3.4). However, as the number of robots is increased, policies computed using time varying graphs do not converge whereas policies trained using a fixed graph still converge to desired behavior. One possible reason for this is that when the number of robots is small, the number of possible graphs is also small and it is easier to learn over this small number of graphs. As the number of robots increase, the number of possible graphs increase exponentially leading to a very large number of graphs that the robot needs to learn over.

To compute the policies, a GNN architecture with L layers is initialized. At each layer according to Eq.6.6, the output is given as:

$$\mathbf{z}^{l+1} = \sigma\big(\mathbf{H}(\mathbf{S})\mathbf{z}^{l}\big) \tag{3.15}$$

where σ is a pointwise non-linear function, $\mathbf{z}^0 = \mathbf{x}_t$ and $\mathbf{z}^L = \Pi = [\pi_1, \ldots, \pi_N]$. In practice, the final layer outputs are parameters of Gaussian distributions from which actions are sampled. Intuitively at every node, the GNN architecture aggregates information and uses

this information to compute policies. In order to satisfy the constraints given in Eq.3.11 and Eq.3.13, we formulate a *centralized* reward structure of the form:

$$r(t) = \begin{cases} -\beta, & \text{if any collisions,} \\ -\sum_i^N E_c(\mathbf{p}_{it}, \mathbf{g}_i) & \text{otherwise} \end{cases} \tag{3.16}$$

Each robot receives the same reward an attempts to learn a policy that best optimizes this reward. It is assumed that the policies for the robots are independent. Thus, the overall loss function for all the robots as given in Eq.6.2

$$J = \sum_{n=1}^{N} \max_{\theta} \mathbb{E}_{\Pi}\left[\sum_{t}^{T} r_t\right] \tag{3.17}$$

where θ now represents the filter weights of the GNN for Π. Consider a trajectory $\tau = (\mathbf{x}_0, \mathbf{a}_0, \ldots, \mathbf{x}_T, \mathbf{a}_T)$. Since the reward along a trajectory is the same for all robots and all robot policies are assumed independent, using direct differentiation the policy gradient is given as :

$$\nabla_\theta J = \mathbb{E}_{\tau\sim(\pi_1,\ldots,\pi_N)}\left[\left(\sum_{t=1}^{T} \nabla_\theta \log[\pi_1(\mathbf{a}_{1t}|\mathbf{x}_{1t}, \mathcal{G}) \ldots \pi_N(\mathbf{a}_{Nt}|\mathbf{x}_{Nt}, \mathcal{G})]\right)\left(\sum_{t=1}^{T} r_t\right)\right] \tag{3.18}$$

For the full algorithm, please see the Algorithm 1. The weights θ, are then updated using any variant of stochastic gradient descent. We call this algorithm Graph Policy Gradients (GPG). In the next section we demonstrate how GPG is able to learn meaningful policies even as the number of robots increase and is also able to transfer filters learned from a small number of robots to a larger number of robots.

Algorithm 2 Graph Policy Gradients

while True **do**

for time $t = [0, \ldots, T]$ **do**

for robots $n = [1, \ldots, N]$ **do**

Record each robots state $\mathbf{x}_t$

Aggregate information at each robot n from its neighbors $\mathcal{B}_n$ according to Eq. 3.3

$$y_n = \sum_{j=n, j \in \mathcal{B}_n} s_{nj} \mathbf{x}_{nt}$$

end for

Collect $\mathbf{x}_t = [\mathbf{x}_{1t}, \ldots, \mathbf{x}_{Nt}]$

Compute K localized graph convolution $\mathbf{z}_t$ as given in Eq.3.4

$$\mathbf{z}_t = \sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{x}_t = \mathbf{H}(\mathbf{S}) \mathbf{x}_t$$

Stack L graph convolution layers such that $\mathbf{z}_t^0 = \mathbf{x}_t$ and $\mathbf{z}_t^L = \Pi$

$$\mathbf{z}_t^{l+1} = \sigma\big(\mathbf{H}(\mathbf{S}) \mathbf{z}_t^l\big)$$

Sample $\mathbf{a}_t = \{\mathbf{a}_{1t}, \ldots, \mathbf{a}_{Nt}\} := \{\pi_1(\mathbf{a}_{1t} | \mathbf{x}_{1t}, \mathcal{G}), \ldots, \pi_N(\mathbf{a}_{Nt} | \mathbf{x}_{Nt}, \mathcal{G})\}$

Execute $\mathbf{a}_t$ and collect reward over all robots $R_t = \sum_{n=1}^{N} r(t)$

end for

Record trajectory $\tau = \big[(\mathbf{x}_0, \mathbf{a}_0, R_0), \ldots, (\mathbf{x}_T, \mathbf{a}_T, R_T)\big]$.

Compute the graph policy gradients as given in Eq 6.10 and update weights θ

end while

3.2 EXPERIMENTS

To investigate the performance of GPG, we design experiments to answer three key questions :

- Can GPG learn policies that achieve desired behavior as the number of robots increase?
- How well do the graph filters learned by GPG transfer to a large number of robots?
- Does GPG work with more complex dynamics and controls?

3.2.1 Training for Point Mass Formation Flying with GPG

We first look to establish if GPG can train for formation flying for simple point mass dynamics as the number of robots are increased. The state of every robot n is its absolute position to its goal and the action a_{nt} at time t for robot n is the change in x and y position in the plane. Such a setting necessitates communication between robots. One could argue that the robots simply have to learn to take actions such that $\mathbf{x}_{nt}$ tends to zero. However, consider the example in Fig 3.3. When the goals are reasonably far away $d_1 \approx d_2$ and each robot can take actions that could cause it to collide with its neighbors. Each robot must

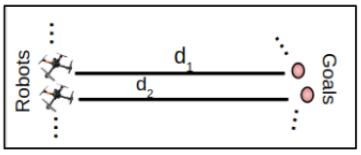


Figure 3.3: Formation Flying.

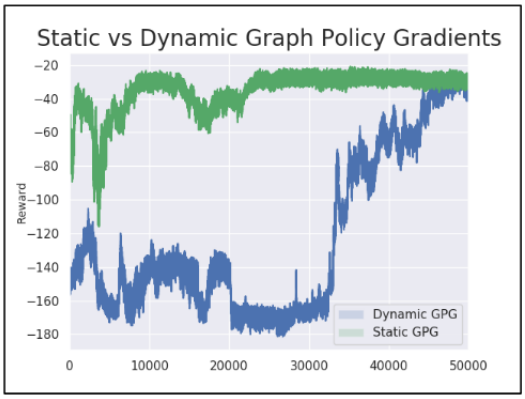


Figure 3.4: 10 robot Formation Flying. Using a static graph during training increases the sample efficiency of GPG. A dynamic graph, i.e a graph that evolves over time as the robots move in space takes longer to converge.

communicate with its neighbors who in turn must communicate with theirs in order to achieve collision free trajectories that take robots to their assigned goals.

To establish relevant baselines, we choose vanilla policy gradients (VPG) that uses the same policy gradient hyperparameters as GPG but differs from GPG in that it uses a 2 layer fully connected network for policy paramterization. We also compare with Proximal Policy Optimization (PPO) [31] a state of the art on policy method for learning continuous policies. In this paper, we employ a batched version of PPO for faster computation [51]. The PPO baseline also employs fully connected networks. Another choice of baseline used in this paper PPO with a recurrent network architecture since the problem on hand requires communication between agents and shared memory is one possible mechanism. The training curves for our experiments with 3,5 and 10 robots can be seen in Fig 3.5. We observe right away that GPG is able to converge in all three cases. VPG does not produce any meaningful behavior and both variants of PPO only produce partial results. In the experiments above, we maintain a static graph $\mathcal{G}$ while training in light of the permutation equivariance property of GNNs. While static graphs are used during training to speed up

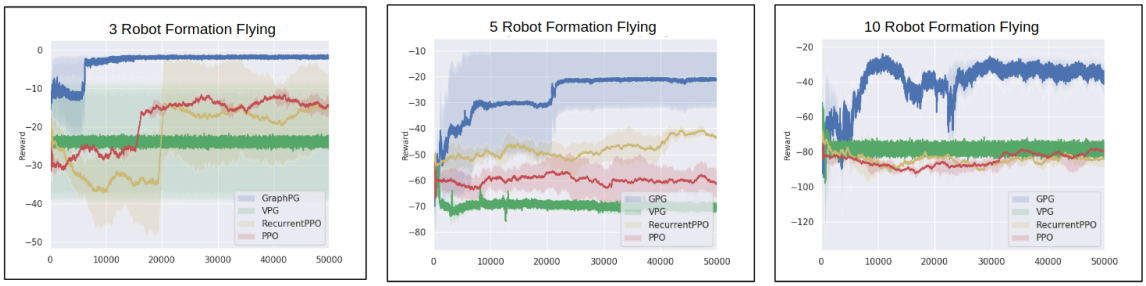


Figure 3.5: Training for Formation Flight. Point mass robots are trained for formation flight. The reward is a centralized reward. Each curve is produced by running three independent runs of the algorithm. Darker line represents mean and shaded area represents mean $\pm$ standard deviation of mean.

the training process, during testing, the graphs are dynamic since the graph convolution yields the same result. We also experiment with a dynamic graph $\mathcal{G}_t$ that evolves as the robots move in space. As hypothesized and guided by the intuition of Theorem 1, using dynamic graphs increases sample complexity of the problem whereas the static graph converges faster. This can be seen in Fig 3.4.

3.2.2 Zero Shot Policy Transfer for Formation Flying

It can be observed from Fig 3.5 that while GPG is able to converge as the number of robots increase, the number of samples required also increases. Training GPG for ten robots alone takes over nine hours on a 12GB NVIDIA 2080Ti GPU whereas we wish to learn policies for hundreds of robots. One possible solution is to realize that the filters trained extract local features and the same filter can be used when working with many more robots. To demonstrate this, we train a filter with a small number of robots (depending on the formation we are interested in) and use the same filter weights without any gradient updates on the larger swarm. Another point to note is that when training for the smaller swarm, we only train for goals that are a small distance away to reduce the number of

samples (it is easier to discover goals that are close by than it is to discover goals much further away) required for training. However, during execution these policies are able to converge even when the goals are hundred units away. A snapshot of some of the formations we can produce using this zero shot transfer can be seen in Fig. 4.2. In Fig. 4.2 (left) and (center), policies are trained on three robots and utilize only 1-hop neighbor information. The figure of eight formation necessitates more communication especially when robots at the edges cross over to form the figure of eight. In Fig. 4.2 (right) the filters are learned by training with five robots with each robot utilizing 3 hop neighbor information.

With these results, it can be concluded that GPG is capable of learning complex behaviors for a small number of robots. These behaviors can then be transferred to larger more complex swarms. To the best of our knowledge there exists no other method to learn continuous control policies that can achieve similar results for so many robots.

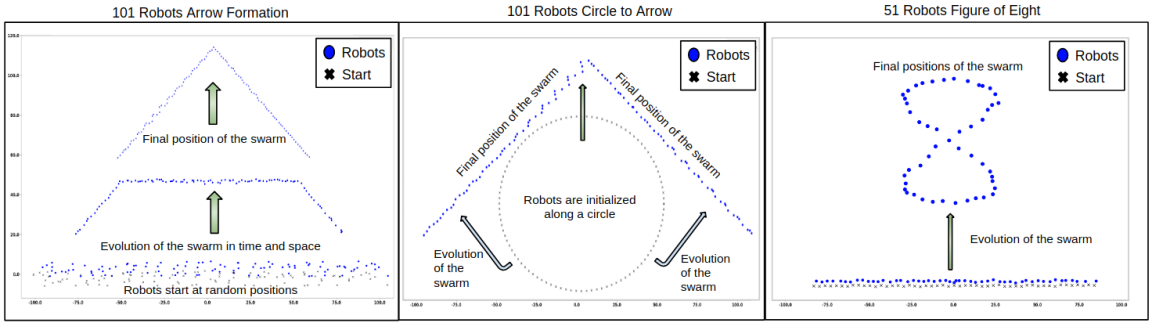


Figure 3.6: Zero Shot Transfer to Large Number of Robots. (Left) Policies are trained for three robots to reach goals that are a small distance away. Robots are randomly initialized in a rectangular region and must reach goals much further than those in the training set. (Center) Robots are initialized on a circle and must execute policies such that the resulting shape is an arrowhead. (Right) Policies trained on swarms of five robots are transferred over to form a figure of eight. The choice of 101/51 robots is arbitrary and is not a limiting threshold.

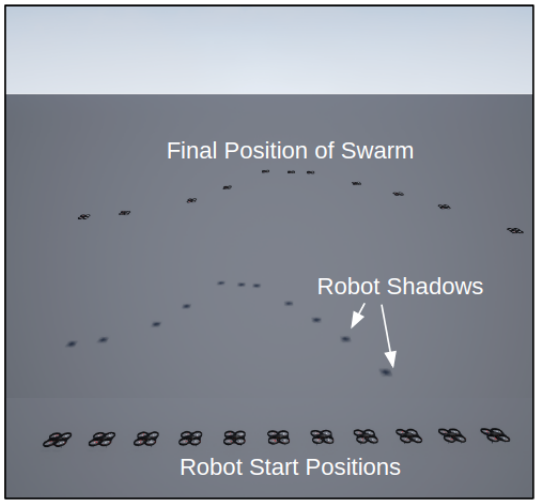


Figure 3.7: 11 robot Arrow Head Formation. The robots are spawned at ground and are manually controlled for takeoff. Once the robots are at a desired height, control is handed over to GPG. (see Appendix for detailed figures)

3.2.3 Complex dynamics and control

In the experiments considered above, actions for robot n are simply the change in position in the plane. Further, it is assumed that full control of robots can be achieved and at every step robots have zero velocities. These assumptions are infeasible in the real world. Thus, we also conduct experiments to test the efficacy of GPG on more realistic dynamics and control.

To simulate this, we use the AirSim simulator introduced in [50]. Robots are now assumed to have finite mass and inertia and obey single integrator dynamics. The state for each robot used by GPG is now given as $\mathbf{x}_{nt} := [(\mathbf{p}_{nt} - \mathbf{g}_n), \dot{\mathbf{p}}_{nt}]$ where $\dot{\mathbf{p}}_{nt}$ is the velocity at time t for robot n. Thus, the state for each robot is not just its relative position but also includes its current velocity. The actions now represent change in velocity in the plane. Similar to the point mass experiments, policies are first learned for a small number of robots in AirSim. In addition to the non simplistic dynamics, the difficulty of training

policies in AirSim is further compounded by the fact that AirSim is a real time simulator and produces an order of magnitude fewer training points as compared to the point-mass simulation over a fixed period of time. Thus, it is even more imperative when working with such a simulator to be able to learn policies for a small number of robots that require fewer samples and then be able to transfer these policies to a larger number of robots for which direct training would otherwise be infeasible. GPG being a model free algorithm faces no additional difficulty in adapting to the single integrator dynamics. A snapshot of results produced using GPG in AirSim can be seen in Fig 3.4. In this case, the baselines used in Section 3.2.1 are unable to learn reasonable behaviors within 50K episodes. This can be attributed to the fact that the observations in AirSim are noisier and the baselines most likely need a lot more samples to converge.

For the experiments in AirSim, we assume the robots to obey single integrator dynamics. Robot n's position in the plane at time t is given as $\mathbf{p}_{nt}$ and its velocity at time t is given

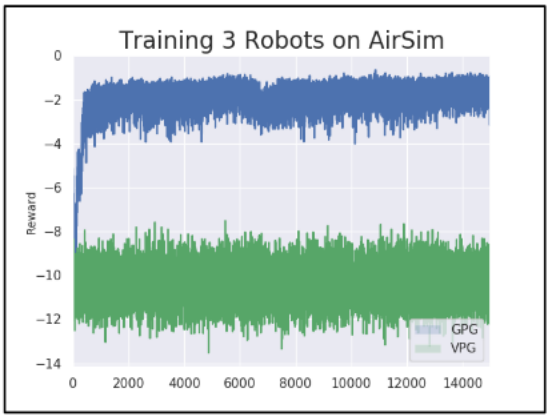


Figure 3.8: Training 3 Robots in AirSim.

as $\dot{\mathbf{p}}_{nt}$. The action $\mathbf{a}_{nt}$ chosen by the robot gives the change in velocities. The state of robot n then evolves according to :

$$\begin{bmatrix} \mathbf{p}_{nt+1} \\ \dot{\mathbf{p}}_{nt+1} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & T_s\mathbf{I} \\ 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{p}_{nt} \\ \dot{\mathbf{p}}_{nt} \end{bmatrix} + \begin{bmatrix} 0 \\ 0.1T_s\mathbf{I} \end{bmatrix} \mathbf{a}_{nt} \tag{3.19}$$

where T_s is sampling time. To speed up training, the velocities are clipped in a range of $[-1, 1]$m/s. The robots are initialized along a straight line with a fixed distance between them. An external controller is used to arm the robots and for takeoff. Once all the robots are at a fixed height, control is handed over to GPG. Further, when the robots reach their goals, they do not enter hover mode. GPG outputs small velocities for the robots and as a result when robots are close to their goals, they tend to oscillate around the goal point.

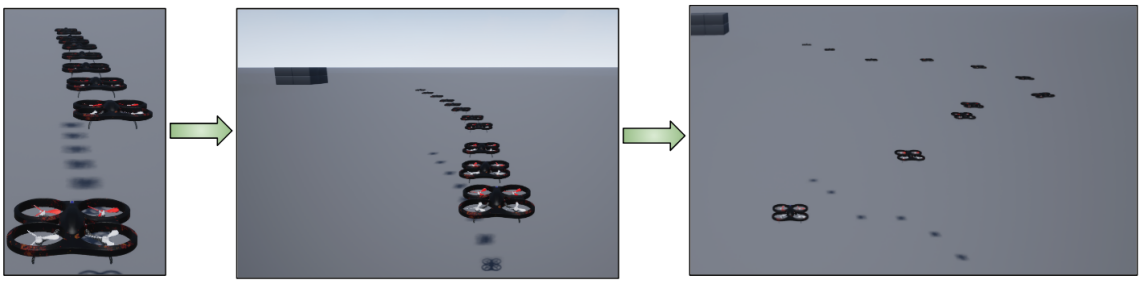


Figure 3.9: Arrowhead Formation Flying for 11 robots in AirSim.

3.3 HYPERPARAMETERS

For the point mass experiments, when learning one-hop neighbor information, we use a 2 layer GNN with a tanh nonlinearity. The input layer consists of 16 hidden units and is fed the state representation $\mathbf{x}_t$. The output of this layer is then fed into a μ layer and a σ

layer both of which have 16 hidden units. Actions for the robots are sampled using these parameters. We use the Deep Graph Library (DGL) to represent our GNNs. The learning rate is set to 1e−3 and uses the Adam optimizer. For the VPG baseline, the learning rate and the optimizer remain the same. The network is parametrized by fully connected layers where the input to the first layer is the state representation and consists of 64 hidden units followed by a ReLU non linearity. The output of this first layer is then fed into separate μ and σ layers each consisting of 64 hidden units and ReLU activation functions. We invite the reader to take a look at the code attached with this paper for more details.

3.4 RELATED WORK

In the recent past, there has been an immense amount of interest in learning policies for a large collection of agents. Literature in multi-agent RL has argued for a centralized training decentralized execution scheme [52, 53] where each agent has its own policy network but during train time, it has access to an additional critic network that co-ordinates information among all agents. Naturally, such a scheme is not scalable as the input size of the critic network grows as the number of robots increase in addition to the inherent complexity of training hundreds of separate policies simultaneously. Another line of thinking in this field, has been the idea to approximate the policies of all the agents using meta-learning [54, 37]. These works while impressive, still run into the problem of having to execute rollouts for a large number of agents to learn a meaningful meta-representation. Closest to our work, perhaps is the work of [55] where the authors propose using CNNs on an encoded graph representation of the agents. In contrast to our work, the policies learned are discrete and the convolutions do not fully exploit the local graph structure.

3.5 DISCUSSION AND GUIDING IDEAS

The problem of learning policies that map state representation to control commands for a large number of robots is immensely challenging, even if the robots are homogeneous. In this work, we propose using the underlying graph structure as additional information. To exploit information from the graph we propose using GNNs to parametrize the policies. Using GNNs we are able to learn local filters defined on the graph that extract local information. The additional benefit of using these local feature extractors is that they can be used to alleviate the problem of needing a large number of rollouts as the number of robots increase as the number of robots increase as demonstrated by the experiments in Section 3.2.2. In the next chapter, we revisit the concurrent goal assignment and planning problem discussed in Chapter 2 and use GPG to design learning based model free solutions that are able to scale to a large number of robots. It might even be possible to add other sensor representations such as camera images or lidar instead of requiring exact positions of the robots in space. These could be first processed using standard architectures such as CNNs and then be fed into GPG to learn policies for a large swarm of robots directly from on board sensors. We shall use this idea to learn perception action loops for robots in Chapter 5.

4 GRAPH POLICY GRADIENTS FOR LARGE SCALE UNLABELED MOTION PLANNING

In Chapter 2 we introduced the unlabeled assignment and planning problem and also introduced a model free multi-agent reinforcement learning solution as an alternative to existing model based solutions. However, a key drawback of these methods was the difficulty in scaling up training for a larger number of robots. In Chapter 3 we introduced Graph Policy Gradients (GPG) as a method to train a large number of robots for a relatively simpler problem of formation flying, where each robot has an assigned goal that it must reach without colliding into its neighbors and all robots must reach their goals in the least amount of time. Through varied experiments, we demonstrated how GPG can adapt to a large swarm by training on a simpler scenario where there the number of robots operating are much smaller than that at inference. The key idea that enabled the use of GPG for the formation flying problem, was the careful design of the state space for the robots such that the designed solution could leverage the permutation invariance property inherent in graph neural networks and ensure that the solution can scale to a larger number of robots at inference time. In this chapter, we extend these ideas to the concurrent assignment and planning problem. We demonstrate that it is easy enough to design a permutation invariant state space representation for the unlabeled motion planning problem such that GPG can be adapted to it. Lastly, through varied simulations and theoretical results, we demonstrate the efficacy of Graph Policy Gradients for the unlabeled motion planning problem.

4.1 DISTRIBUTED COLLABORATIVE UNLABELED MOTION PLANNING

We consider the problem of navigating N disk-shaped robots to a set of N goals of radius R. We say that the navigation problem is unlabeled because there is no prior matching between robots and goals. Any robot can cover any goal. The positions of the goals are fixed and we use $\mathbf{g}_n$ to denote the position of goal n. The positions of robots change over discrete time index t and are denoted as $\mathbf{r}_n(t)$ for robot n. We stack goal positions into the matrix $\mathbf{G} = [\mathbf{g}_1^T; \dots; \mathbf{g}_N^T]$ whose nth row is the position of goal n and robot positions into the matrix $\mathbf{R}(t) = [\mathbf{r}_1^T(t); \dots; \mathbf{r}_N^T(t)]$ whose nth row is the position of robot n. We further group these two matrices to define the system state

$$\mathcal{S}(t) = \langle \mathbf{R}(t), \mathbf{G} \rangle = \langle [\mathbf{r}_1^T(t); \dots; \mathbf{r}_N(t)^T], [\mathbf{g}_1^T; \dots; \mathbf{g}_N^T] \rangle \tag{4.1}$$

We assume that robot positions evolve according to a choice of action $\mathbf{a}_n(t) \in \mathcal{A}$ as determined by some stationary distribution with conditional density $p(\mathbf{r}_n(t+1)|\mathbf{r}_n(t), \mathcal{S}_n(t))$. This first order dynamical model implies that the state $\mathcal{S}(t)$ is a complete description of the system at time t. We further assume operation in open space. Both these choices are for simplicity of exposition. Incorporating obstacles (Section 4.4) and high order complex dynamics is an easy extension.

The goal of the team of robots is to cover all goals in the sense that all goals $\mathbf{g}_n$ have at least one robot within distance R of its location. To measure how well the configuration $\mathcal{S}(t)$ in (4.1) accomplish this task, the reward $r(\mathcal{S}(t))$ counts the number of goals covered.

This is equivalent to counting all the goals for which $\|\mathbf{g}_n - \mathbf{r}_m\| \leqslant R$ for at least one robot m, which we can write as the sum

$$r(\mathcal{S}(t)) = \sum_{n=1}^{N} \mathbb{I}\Big[\min_{m} \|\mathbf{g}_n - \mathbf{r}_m(t)\| \leqslant R\Big] \tag{4.2}$$

In (4.2) only the robots that are within distance R of a goal contribute to the reward. If there are several robots within R of a goal, only the closest contributes toward increasing $r(\mathcal{S}(t))$. The maximum reward $r(\mathcal{S}(t)) = N$ occurs when all goals are covered by one robot. Therefore, the necessary and sufficient condition for all goals to be covered at some time t is for the reward to attain its maximum $r(\mathcal{S}(t)) = N$.

Our interest here is designing policies to reach the maximum reward $r(\mathcal{S}(t)) = N$ (Section 4.2) with a distributed collaborative system. The system is distributed because agents have access to local state information only and collaborative because nearby agents communicate with each other. To describe the locality of information and communication, consider orderings of goals and robots relative to their distance to a given robot. Formally, let $\mathbf{r}_{n[m]}(t)$ be the mth closest robot to robot n other than n itself so that for all $1 \leqslant m \leqslant m' \leqslant N-1$ it holds

$$\|\mathbf{r}_{n[m]}(t) - \mathbf{r}_n(t)\| \;\leqslant\; \|\mathbf{r}_{n[m']}(t) - \mathbf{r}_n(t)\|. \tag{4.3}$$

Likewise, let $\mathbf{g}_{n[m]}$ be the mth closest goal to robot n so that for any $1 \leqslant m \leqslant m' \leqslant N$ we can write

$$\|\mathbf{g}_{n[m]}(t) - \mathbf{r}_n(t)\| \;\leqslant\; \|\mathbf{g}_{n[m']}(t) - \mathbf{r}_n(t)\|. \tag{4.4}$$

Observe that the position $\mathbf{g}_{n[m]}(t)$ of the mth closest goal changes over time as robot n moves through the environment.

We assume that each agent senses *at most* the M goals and at most the M robots that are closest to its location and that it can communicate with the M closest robots. We therefore define the state of robot n at time t as a row vector concatenating its own location, the location of its Mth closest robots, and the location of its Mth closest goals,

$$\mathbf{x}_n(t) = \Big[\mathbf{r}_n^T(t);\ \mathbf{r}_{n[1]}^T(t); \dots; \mathbf{r}_{n[M]}^T(t); \\ \mathbf{g}_{n[1]}^T(t); \dots; \mathbf{g}_{n[M]}^T(t)\Big]. \tag{4.5}$$

For future reference we introduce d to denote the number of entries of $\mathbf{x}_n(t)$. This number is $d = 2(2M+1)$ for planar navigation and $d = 3(2M+1)$ for navigation in three dimensions. Communication between agents is also restricted to local information exchanges. We model this with a communication graph whose adjacency matrix we denote by $\mathbf{S}(t) \in \mathbb{R}^{N \times N}$. Entries of this matrix $S_{nm}(t)$ are binary and are 1 only if robot m is among the Mth closest robots to robot n. We write this formally as

$$S_{nm}(t) = \mathbb{I}\Big[\mathbf{r}_m \in \big\{\mathbf{r}_n(t), \mathbf{r}_{n[1]}(t), \dots, \mathbf{r}_{n[M]}(t)\big\}\Big] \tag{4.6}$$

where we point out that we also add self loops to $\mathbf{S}$ because we have made $S_{nn} = 1$. Observe that M limits the number of robots that communicate with each robot n, as well as the number of robots and goals sensed by each robot n. These three parameters can be distinct in practical implementations but we make them equal here to simplify notation.

Robots are given access to their local states $\mathbf{x}_n(t)$ as defined in (4.5) and can exchange information with neighboring agents as dictated by the graph $\mathbf{S}(t)$ whose entries are given by (4.6). Given their local states and information received from neighbors, robot n chooses an e-dimensional action $\mathbf{a}_n(t) \in \mathbb{R}^e$ that controls the transition into position $\mathbf{r}_n(t+1)$

according to the dynamical model $p(\mathbf{r}_n(t+1)|\mathbf{r}_n(t), \mathbf{a}_n(t))$. We write these stochastic policies as:

$$\mathbf{a}_n(t) = \pi_n\big(\mathbf{a}_n(t) \,\big|\, \mathbf{x}_n(t); \mathbf{S}(t)\big). \tag{4.7}$$

The notation in (4.7) signifies that $\mathbf{S}_n(t)$ is chosen according to the local state $\mathbf{x}_n(t)$ – i.e., the policy is distributed – and information exchanges with neighboring nodes as dictated by the graph $\mathbf{S}(t)$ – i.e., the policy is collaborative. We emphasize that the graph $\mathbf{S}(t)$ affects the choice of action because it limits the information accessible to robot n and that it therefore is an important component of the robots policy. Notwithstanding, these graph is not necessarily known to robot n and our policies will be executable without this knowledge (Section 4.2). The individual distributed collaborative policies in (4.7) generate the product policy $\Pi = \pi\big(\mathbf{a}_1(t) \,\big|\, \mathbf{x}_1(t); \mathbf{S}(t)\big) \times \ldots \times \pi\big(\mathbf{a}_N(t) \,\big|\, \mathbf{x}_N(t); \mathbf{S}(t)\big)$. We want to jointly optimize over individual policies to maximize the discounted reward

$$J(\Pi) = \max_{\Pi} \mathbb{E}_{\Pi}\left[\sum_{t}^{T} \gamma^t r(\mathcal{S}(t))\right] \tag{4.8}$$

where the reward $r(\mathcal{S}(t))$ is as given in (4.2). Our objective in this paper is to learn policies having the form in (4.7) that maximize the discounted reward in (4.8). We do so with the Graph Policy Gradient method that we introduce next.

4.2 GRAPH POLICY GRADIENTS FOR DISTRIBUTIVE UNLABELED MOTION PLANNING

To find the policy Π that maximizes the reward $J(\Pi)$ in (4.17) it is customary to introduce a policy parameterization. Our key technical innovation is to parameterize Π with a graph neural network (GNN), which as we will explain in Section 4.3 not only respects but also leverages the locality of the distributed collaborative motion planning problem we discussed in Section 4.1. As desciribed in Chapter 3 GNNs are generalizations of convolutional neural networks (CNNs) built on the definition of convolutional graph filters. Here, we add some more formalization to our definition of a GNN to fit into the unlabeled motion planning paradigm. Formally, define the state matrix $\mathbf{X}(t) \in \mathbb{R}^{n \times d}$ whose nth row is the state of robot n [cf. (4.5)],

$$\mathbf{X}(t) = [\mathbf{x}_1(t); \dots; \mathbf{x}_N(t)] \in \mathbb{R}^{N \times d}. \tag{4.9}$$

A graph convolutional filter to process $\mathbf{X}(t)$ on the graph $\mathbf{S}(t)$ is defined by a set of K coefficient matrices $\mathbf{H}_{1k} \in \mathbb{R}^{d \times d_1}$. These matrices serve as coefficients of a polynomial on $\mathbf{S}(t)$ that operates on the state $\mathbf{X}(t)$ to produce the output $\mathbf{Y}(t) \in \mathbb{R}^{d_1}$ given by

$$\mathbf{Z}_1(t) = \sum_{k=0}^{K-1} \mathbf{S}^k(t)\mathbf{X}(t)\mathbf{H}_{1k}. \tag{4.10}$$

The output of the graph filter in 4.10 is further processed with a pointwise nonlinearity to produce the layer 1 output signal

$$\mathbf{X}_1(t) = \sigma\Big[\, \mathbf{Z}_1(t) \,\Big] = \sigma\Big[\sum_{k=0}^{K-1} \mathbf{S}^k(t)\mathbf{X}(t)\mathbf{H}_{1k} \Big]. \tag{4.11}$$

In a GNN with multiple layers the signal $\mathbf{X}_1(t)$ is further processed with a graph filter with coefficients $\mathbf{H}_{1k} \in \mathbb{R}^{d_1 \times d_2}$ and a pointwise nonlinearity, to produce the output of Layer 2. In general, there are a total of L layers each of which is defined by a set of K coefficients $\mathbf{H}_{1k} \in \mathbb{R}^{d_{l-1} \times d_l}$ which produce output signals $\mathbf{X}_l(t)$ according to the recursion

$$\mathbf{X}_l(t) = \sigma\Big[\ \mathbf{Z}_l(t)\ \Big] = \sigma\Big[\sum_{k=0}^{K-1} \mathbf{S}^k(t)\mathbf{X}_{l-1}(t)\mathbf{H}_{lk}\Big]. \tag{4.12}$$

The output of the Lth layer is the output of the GNN and we propose here to use it as the mechanism for generating robot actions. Specifically, stack the individual actions of all agent into the action matrix $\mathbf{A}(t) = [\mathbf{a}_1^T(t); \dots; \mathbf{a}_N^T(t)] \in \mathbb{R}^{n \times e}$ and make

$$\mathbf{A}(t) = \mathbf{X}_L(t) = \Phi(\mathbf{X}(t), \mathbf{S}(t); \mathbf{H}). \tag{4.13}$$

Notice that in (4.13) we have introduced the notation $\Phi(\mathbf{X}(t), \mathbf{S}(t); \mathbf{H})$ to represent the GNN's output. This output depends on the state input $\mathbf{X}(t)$, the graph $\mathbf{S}(t)$ and the filter tensor $\mathbf{H} := \{\mathbf{H}_{lk}\}_{l,k}$ that groups the coefficient matrices of all layers and all orders. The advantage of using a GNN to parameterize actions $\mathbf{A}(t)$ is that they respect the structure of the distributed collaborative policies in (4.7). The graph filters in (4.10) and (4.12) are made up of diffusion operations that involve interactions between neighbors only. Consider the product $\mathbf{S}(t)\mathbf{X}(t)$ and observe that the sparsity pattern of $\mathbf{S}(t)$ is such that the nth row of this product is given by

$$[\mathbf{S}(t)\mathbf{X}(t)]_n = \sum_{m:S_{mn}=1} \mathbf{x}_m(t) \tag{4.14}$$

If we interpret the row $\big[\mathbf{A}(t)\mathbf{X}(t)\big]_n$ as a quantity that is evaluated by robot n, it follows that robot n can evaluate this row by communicating with neighboring nodes only.

Subsequent entries in the graph filters in (4.10) can be recursively evaluated noticing that $\mathbf{S}^k(t)\mathbf{X}(t) = \mathbf{S}^{k-1}(t)\mathbf{X}(t)$ and that we can therefore write:

$$[\mathbf{S}^k(t)\mathbf{X}(t)]_n = \sum_{m:S_{mn}=1} [\mathbf{S}^{k-1}(t)\mathbf{X}(t)]_m \tag{4.15}$$

It follows that all of the summands in (4.10) can be computed exclusively through local information exchanges. The *pointwise* nonlinearity in (4.11) can also be locally implemented. Subsequent layers can be evaluated in a distributed manner as well, because the argument in (4.14) and (4.15) is not specific to Layer 1. Through this recursive distributed computations, robot n ends up computing the nth row of the GNN output $[\mathbf{X}_L(t)]_n = [\Phi(\mathbf{X}(t), \mathbf{A}(t); \mathbf{H})]_n$. This row is used to define the local policy in (4.7) as

$$\mathbf{a}_n(t) = [\mathbf{A}(t)]_n = [\mathbf{X}_L(t)]_n = [\Phi(\mathbf{X}(t), \mathbf{S}(t); \mathbf{H})]_n \tag{4.16}$$

The graph policy gradient method (GPG) is the search for a joint policy Π that maximizes the cost in (4.17) over the space of policies $\Pi(\mathbf{H})$ that produce actions parameterized with a GNN of the form in (4.12). The solution of this optimization is the optimal filter tensor

$$\mathbf{H}^* = \arg\max_{\mathbf{H}} \mathbb{E}_{\Pi(\mathbf{H})}\left[\sum_t^T \gamma^t r(\mathcal{S}(t))\right]$$
$$\Pi(\mathbf{H}) : \mathbf{A}(t) = \Phi(\mathbf{X}(t), \mathbf{S}(t); \mathbf{H}). \tag{4.17}$$

In (4.17), the filter tensor $\mathbf{H}$ along with the state $\mathbf{X}(t)$ and the graph $\mathbf{S}(t)$ determine the choice of action $\mathbf{A}(t)$. This joint action controls the state transition as dictated by the dynamical model $p(\mathbf{r}_n(t+1)|\mathbf{r}_n(t), \mathbf{S}_n(t))$. We want to find a filter tensor $\mathbf{H}$ that results in the maximum reward $J(\Pi(\mathbf{H}))$.

The optimal filter can be computed through online policy gradient methods. Robots roll out trajectories $\tau \sim \Pi$ and collect rewards $r(\mathcal{S}(t))$ over a time horizon $t = 0, \dots, T$. Since the policies are assumed to be independent, the policy gradient wrt $\mathbf{H}$, $\nabla_{\mathbf{H}} J$ can be given as:

$$= \mathbb{E}_{\tau \sim (\Pi)} \left[\Big(\sum_{t=1}^{T} \nabla_{\mathbf{H}} \log[\pi_1 \times \ldots \times \pi_N] \Big) \Big(\sum_{t=1}^{T} r(\mathcal{S}(t)) \Big) \right] \tag{4.18}$$

In parameterizing actions with a GNN we ensure the possibility of having a distributed implementation. We will see in the upcoming Section 4.3 that GNNs also exhibit an invariance to the labeling that substantiates transferability properties that we explore in the numerical experiments in Section 4.4.

4.3 PERMUTATION INVARIANCE OF GNN POLICY PARAMETERIZATIONS

Let the GNN coefficients $\mathbf{H}^*$ be the solution that maximizes (4.17) for a given $\mathbf{S}(t)$ and $\mathbf{X}(t)$. Consider another set of graphs $\tilde{\mathbf{S}}(t)$ and state vectors $\tilde{\mathbf{X}}(t)$ that are produced by permuting $\mathbf{S}(t)$ and $\mathbf{X}(t)$. Let the filter coefficients after training on $\tilde{\mathbf{S}}(t)$ and $\mathbf{X}(\tilde{t})$ be $\tilde{\mathbf{H}}^*$. To study the relationship between $\mathbf{H}^*$ and $\tilde{\mathbf{H}}^*$ we first define a set of permutation matrices of dimension N such that $\mathcal{P} = \{\mathbf{P} \in \{0,1\}^{N \times N} \ \mathbf{P}\mathbf{1} = \mathbf{1}, \mathbf{P}^\top \mathbf{1} = \mathbf{1}\}$

Such a permutation matrix $\mathbf{P}$ is one for which the product $\mathbf{P}^\top \mathbf{X}(t)$ reorders the entries of any $\mathbf{X}(t)$ and the operation $\mathbf{P}^\top \mathbf{S}(t)\mathbf{P}$ produces a reordering of the rows and columns of

any given $\mathbf{S}(t)$. The policy for system with $\mathbf{S}(t)$ is $\Pi = \Phi(\mathbf{X}(t), \mathbf{S}(t); \mathbf{H})$. Analogously, let the policy for the permuted system be given as:

$$\tilde{\Pi} = \Phi(\tilde{\mathbf{X}}(t), \tilde{\mathbf{S}}(t); \tilde{\mathbf{H}}) \text{ where } \tilde{\mathbf{S}}(t) = \mathbf{P}^\top \mathbf{S}(t)\mathbf{P}, \tag{4.19}$$

$\tilde{\mathbf{X}}$ is the permuted state of all robots and $\tilde{\mathbf{H}}$ is the vector of filter coefficients that parameterizes $\tilde{\Pi}$. This leads to:

Theorem 2. *Given a configuration of robots represented by* $\mathbf{X}(t)$ *that define an underlying graph* $\mathcal{G}$ *with graph shift operator* $\mathbf{S}(t)$ *and another configuration of robots given by* $\tilde{\mathbf{X}}(t)$ *and* $\tilde{\mathbf{S}}(t)$ *then the graph filter coefficients* $\mathbf{H}^*$ *and* $\tilde{\mathbf{H}}^*$ *which are the optimal solutions for the systems* $(\mathbf{S}(t), \mathbf{X}(t))$ *and* $(\tilde{\mathbf{S}}(t), \tilde{\mathbf{X}}(t))$ *respectively are equivalent,*

$$\mathbf{H}^* \equiv \tilde{\mathbf{H}}^* \tag{4.20}$$

In order to prove Theorem 2, we must prove that both the unlabeled motion planning problem and the GNNs parameterized by $\tilde{\mathbf{H}}^*$ and $\mathbf{H}^*$ are permutation equivariant.

4.3.1 Equivariance of GNNs for Unlabeled Motion Planning

Proposition 1. Given a system of robots and goals $\mathcal{S}(t) = \{\mathbf{r}(t), \mathbf{g}\}$ and another system where the robot states are permuted $\tilde{\mathcal{S}}(t) = (\tilde{\mathbf{r}}(t), \mathbf{g})$ where $\tilde{\mathbf{r}}(t) = \mathbf{P}^\top \mathbf{r}(t)$, the corresponding robot states $\mathbf{X}(t) = [\mathbf{x}_1(t), \ldots, \mathbf{x}_N(t)]^\top$ and $\tilde{\mathbf{X}}(t) = [\tilde{\mathbf{x}}_1(t), \ldots, \tilde{\mathbf{x}}_N(t)]^\top$ are permutation related, i.e $\tilde{\mathbf{X}}(t) = \mathbf{P}^\top \mathbf{X}(t)$.

Proof This is true because we have used ordering to construct the local robot states in (4.5) and orderings are invariant to permutations. Formally, consider indexes n and $\tilde{n}$ with $[\mathbf{P}]_{n\tilde{n}} = 1$. It follows from $\tilde{\mathbf{R}}(t) = \mathbf{P}^\top \mathbf{R}(t)$ that $\mathbf{r}_n(t) = \mathbf{r}_{\tilde{n}}(t)$; which means that robot n has

been mapped to robot $\tilde{n}$ in the permutation. Since orderings are independent of labeling it follows that for all robots we must have $\mathbf{r}_{n[m]}(t) = \mathbf{r}_{\tilde{n}[m]}(t)$ [cf. (4.3)]. Likewise, for all goals we must have $\mathbf{g}_{n[m]}(t) = \mathbf{g}_{\tilde{n}[m]}(t)$ [cf. (4.4)]. Given the definition of the state row vector in (4.5) we therefore have that $\mathbf{x}_n(t) = \mathbf{x}_{\tilde{n}}(t)$. Further recalling the definition of the state matrix $\mathbf{X}(t)$ in (4.9) and the assumption that $[\mathbf{P}]_{n\tilde{n}} = 1$ we also have that $\mathbf{x}_n(t) = [\mathbf{P}^T\mathbf{X}(t)]_{\tilde{n}}$. Putting the latter two statements together we conclude that $[\mathbf{P}^T\mathbf{X}(t)]_{\tilde{n}} = \mathbf{x}_{\tilde{n}}(t)$. Since this is true for arbitrary $\tilde{n}$ we must have $\mathbf{P}^T\mathbf{X}(t) = \tilde{\mathbf{X}}(t)$. In order to show that the GNN parameterization is equivariant for both settings, consider the following proposition:

Proposition 2. Given robots with states $\mathbf{X}(t)$ and $\tilde{\mathbf{X}}(t)$ and underlying graphs $\mathbf{S}(t)$ and $\tilde{\mathbf{S}}(t)$ such that $\tilde{\mathbf{X}}(t) = \mathbf{P}^\top\mathbf{X}(t)$ (from Proposition 1) and $\tilde{\mathbf{S}}(t) = \mathbf{P}^\top\mathbf{S}(t)\mathbf{P}$ for some permutation matrix $\mathbf{P}$, the outputs of a GNN policy Φ with filter coefficients $\mathbf{H}$ to the pairs $(\mathbf{S}(t), \mathbf{X}(t))$ and $(\tilde{\mathbf{S}}(t), \tilde{\mathbf{X}}(t))$ are related by:

$$\Phi(\tilde{\mathbf{X}}(t), \tilde{\mathbf{S}}(t); \mathbf{H}) = \mathbf{P}^\top\Phi(\mathbf{X}(t), \mathbf{S}(t); \mathbf{H}) \tag{4.21}$$

Proof The output of the GNN filter for the system $(\tilde{\mathbf{S}}(t), \tilde{\mathbf{X}}(t))$ as given in (4.12) is:

$$\Phi(\tilde{\mathbf{X}}(t), \tilde{\mathbf{S}}(t); \mathbf{H}) = \sum_{k=0}^{K} h_k\tilde{\mathbf{S}}(t)^k\tilde{\mathbf{X}}(t) \tag{4.22}$$

This can be expressed as:

$$\sum_{k=0}^{K} h_k\tilde{\mathbf{S}}(t)^k\tilde{\mathbf{X}} = \sum_{k=0}^{\infty} h_k(\mathbf{P}^\top\mathbf{S}(t)^k\mathbf{P})\mathbf{P}^\top\mathbf{X}(t) \tag{4.23}$$

Using the fact that $\mathbf{P}$ is an orthogonal matrix which in turn implies that $\mathbf{P}^\top\mathbf{P} = \mathbf{P}\mathbf{P}^\top = \mathbf{I}$:

$$\sum_{k=0}^{\infty} h_k(\mathbf{P}^\top\mathbf{S}(t)^k\mathbf{P})\mathbf{P}^\top\mathbf{X}(t) = \mathbf{P}^\top\sum_{k=0}^{\infty} h_k\mathbf{S}(t)^k\mathbf{X}(t)$$
$$= \mathbf{P}^\top\Phi(\mathbf{X}(t), \mathbf{S}(t); \mathbf{H}) \tag{4.24}$$

Thus, proving Proposition 2. Intuitively, Proposition 2 tells us that reordering the robots states and corresponding nodes in the graph representation which is fed into the GNN policy, results in an appropriate reordering of the outputs of the filter without any change in the weights of the policy.

4.3.2 Equivariance of Unlabeled Motion Planning

In order to show that the unlabeled motion planning is permutation equivariant, we look at the cost J.

Proposition 3. The reward function J is permutation equivariant. i.e for all permutation matrices $\mathbf{P} \in \mathcal{P}$, $J_\Pi = J_{\tilde{\Pi}}$ where $J_\Pi = \max_\Pi \mathbb{E}_\Pi\left[\sum_t^T \gamma^t \sum_{n=1}^N \mathbb{I}\left[\min_m \|\mathbf{g}_n - \mathbf{r}_m\| \leqslant R\right]\right]$

Proof For the permuted system the reward is given as :

$$r(\tilde{S}(t)) = \sum_{n=1}^{N} \mathbb{I}\left[\min_m \|\mathbf{g}_n - \tilde{\mathbf{r}}_m\| \leqslant R\right] \tag{4.25}$$

Since the set of robots $\mathbf{r}$ and $\tilde{\mathbf{r}}$ are the same:

$$\tilde{\mathbf{r}}_{i[1]} = \arg\min_j \|\mathbf{g}_i - \tilde{\mathbf{r}}_j\| = \mathbf{r}_{i[1]} \tag{4.26}$$

Which leads us to $r(\mathcal{S}(t)) = r(\tilde{\mathcal{S}}(t))$ and therefore $\sum_t^T r(\mathcal{S}(t)) = \sum_t^T r(\tilde{\mathcal{S}}(t))$. Now,

$$J_{\tilde{\Pi}} = \sum_{n=1}^{N} \max_{\theta} \mathbb{E}_{\tilde{\Pi}} \left[\sum_t^T r(\tilde{\mathcal{S}}(t)) \right] \tag{4.27}$$

$$= \sum_{n=1}^{N} \max_{\theta} \int \sum_t^T r(\tilde{\mathcal{S}}(t)) d\tilde{\Pi} \tag{4.28}$$

Using the fact that $d\tilde{\Pi} = \mathbf{P}^\top d\Pi$ (from Proposition 2) and $r(\mathcal{S}(t)) = r(\tilde{\mathcal{S}}(t))$ we get;

$$\sum_{n=1}^{N} \max_{\theta} \int \sum_t^T r(\tilde{\mathcal{S}}(t)) d\tilde{\Pi} = \sum_{n=1}^{N} \max_{\theta} \mathbf{P}^\top \int \sum_t^T r(\mathcal{S}(t)) d\Pi \tag{4.29}$$

Since the set of robots is $\mathbf{N}$ for both J_Π and $J_{\tilde{\Pi}}$, we can conclude $J_{\tilde{\Pi}} = J_\Pi$ Thus, we conclude that the unlabeled motion planning problem is permutation equivariant. With these results, we construct the following proof:

Proof for Theorem 2 Let optimal coefficients $\mathbf{H}^*$ and $\tilde{\mathbf{H}}^*$ induce a optimal reward J^* and $J^*_{\tilde{\Pi}}$ respectively. Consider the optimal filter coefficient $\mathbf{H}$. From proposition 2, we have:

$$\Phi(\tilde{\mathbf{X}}(t), \tilde{\mathbf{S}}(t); \mathbf{H}) = \mathbf{P}^\top \Phi(\mathbf{X}(t), \mathbf{S}(t); \mathbf{H}) \tag{4.30}$$

$\Phi(\tilde{\mathbf{X}}(t), \tilde{\mathbf{S}}(t); \mathbf{H})$ induces a reward $J_{\tilde{\Pi}}$ while $\mathbf{P}^\top \Phi(\mathbf{X}(t), \mathbf{S}(t); \mathbf{H})$ induces the optimal reward J^*_Π. However, as a result of Proposition 3, we have:

$$J_{\tilde{\Pi}} = J^*_\Pi \tag{4.31}$$

Similarly by considering the optimal filter coefficient $\tilde{\mathbf{H}}^*$:

$$J^*_{\tilde{\Pi}} = J_\Pi \tag{4.32}$$

If the cost J^*_{Π} and $J^*_{\tilde{\Pi}}$ are the optimal costs, then:

$$J^*_{\Pi} \geqslant J_{\Pi} = J_{\tilde{\Pi}} \tag{4.33}$$

$$J^*_{\tilde{\Pi}} \geqslant J_{\tilde{\Pi}} = J^*_{\Pi} \tag{4.34}$$

where the second equality follows from (4.31) and (4.32). From (4.33) and (4.34), we can conclude that

$$J^*_{\tilde{\Pi}} = J^*_{\Pi} \tag{4.35}$$

Thus, we can conclude $\mathbf{H}^* \equiv \tilde{\mathbf{H}}^*$, completing the proof for Theorem 2. One of the bottlenecks of RL for multi-robot systems is the fact that it lacks the ability to scale to a large number of robots. However, Theorem 1 and Proposition 2 offer us a valuable tool. When training, we train only for a small number of robots with graph $\mathbf{S}(t)$ which yields graph filters $\mathbf{H}$. By realizing that the bigger swarm consists of smaller swarms that are permutations $\mathbf{S}(t)$ and the filter coefficients for both are equivalent, we simply reuse the filter $\mathbf{H}$ on each of the smaller swarms. Thus, we are able to achieve large scale unlabeled motion planning without the computational burden of having to train a very large number of robots.

4.4 EXPERIMENTS

To test the efficacy of GPG on the unlabelled motion planning problem, we setup a few experiments in simulation. We establish five main experiments. **1)** Unlabelled motion planning with three, five and ten robots where robots obey point mass dynamics. **2)** GPG is tested on three, five and ten robots but here robots obey single integrator dynamics. **3)** The robots follow single integrator dynamics and additionally the environment is

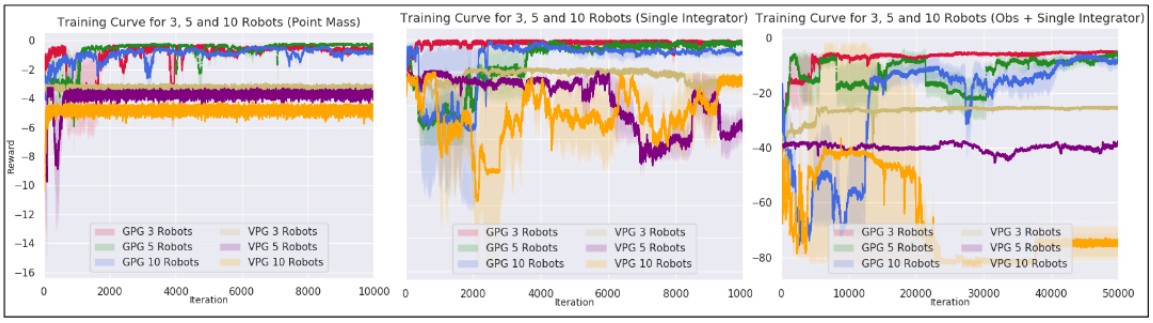


Figure 4.1: Training Curves for 3, 5 and 10 robots Policies trained by GPG are able to converge on experiments with point mass robots, experiments where robots follow single integrator dynamics and are velocity controlled as well as experiments when disk shaped obstacles are present in the environment.

populated with disk shaped obstacles. **4)** The performance of GPG is tested against a model based provably optimal centralized solution for the unlabelled motion planning problem. We demonstrate empirically that the performance of GPG is almost always within a small margin of that of the model based method but comes with the additional advantage of being decentralized. **5)** The learning formulation for the unlabeled motion planning problem is agnostic to underlying robot dynamics or operating conditions in the environment. To demonstrate this, we look to execute the unlabeled motion planning problem in the higher order dynamics simulator AirSim [50] equipped with quadrotors that model real world effects such as downwash and wind. To establish relevant baselines, we compare GPG with Vanilla Policy Gradients (VPG) where the policies for the robots are parameterized by fully connected networks (FCNs). Apart from the choice of policy parameterization there are no other significant differences between GPG and VPG. For GPG, we setup a L-layer GNN. For experiments involving 3 and 5 robots with point mass experiments we find a 2 layer GNN, i,e a GNN that aggregates information from neighbors that are at most 2 hops away to be adequate. For experiments with 10 robots we find that GNNs with 4 layers work the best. For the baseline VPG experiments, we use with 2-4 layers of FCNs. The maximum episode length is 200 steps and the discount factor $\gamma = 0.95$.

In experiments with 3 robots, each robot senses 2 nearest goals and 1 nearest robot. In experiments with 5 and more robots, robots sense 2 nearest goals and 2 nearest robots. The graph connects robots to their 1,2 and 3 nearest neighbors in experiments with 3, 5 and 10 robots respectively.

The behavior of GPG v/s VPG during training can be observed from Fig. 4.1. We observe that in all cases GPG is able to produce policies that converge close to the maximum possible reward (in all three cases maximum possible reward is zero). When compared to the convergence plots of Chapter 3 where we first proposed use of RL for the unlabelled motion planning problem, this represents a large improvement on just training. It can also be observed that GPG converges when robot dynamics are changed or obstacles are added to the environment. While this is not necessarily the optimal solution to the unlabelled motion problem, it is an approximate solution to the unlabelled motion planning problem. The FCN policies represented by VPG in Fig. 4.1 fail to converge even on the simplest experiments.

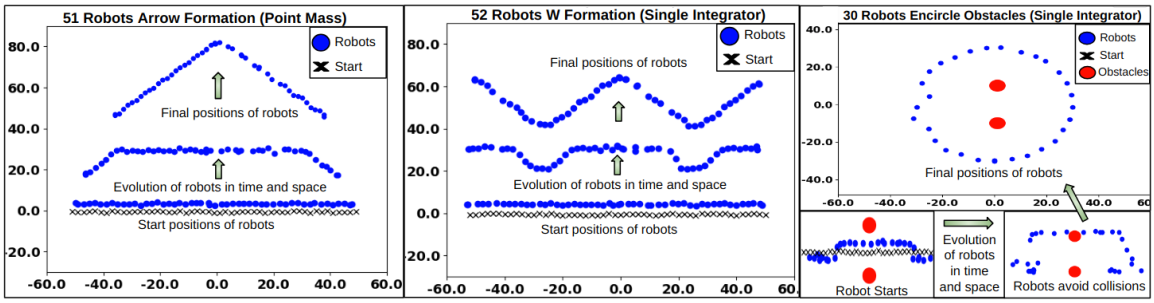


Figure 4.2: Transferring Learned GPG Filters for Large Scale Unlabelled Motion Planning. (Left) A small number of robots are trained to cover goals and follow point mass dynamics. During testing the number of robots as well as the distance of the goals from the start positions are much greater than those seen during training. (Center) A similar experiment is performed but now robots have single integrator dynamics. (Right) In this experiment in addition to single integrator dynamics, the environment also has obstacles that robots must avoid.

4.4.1 Experimental Results - Inference

The previous section shows the feasibility of GPG as a method for training a large swarm of robots to approximately solve the unlabelled motion planning problem. However, training a large number of robots is still a bottleneck due to the randomness in the system. Training 10 robots with simple dynamics on a state of the art NVIDIA 2080 Ti GPU with a 30 thread processor needs 7-8 hours. We see from our experiments that this time grows exponentially with an increase in the number of robots. Thus, to overcome this hurdle and to truly achieve large scale solutions for the unlabelled motion planning problem, we hypothesize that since the graph filters learned by GPG only operate on local information, in a larger swarm one can simply slide the same graph filter everywhere in the swarm to compute policies for all robots without any extra training. Intuitively, this can be attributed to the fact that while the topology of the graph does not change from training time to inference time, the size of the filter remains the same. As stated before, this is akin to sliding a CNN filter on a larger image to extract local features after training it on small images. To demonstrate the effect of GPG during inference time, we setup three simple experiments where we distribute goals along interesting formations. As described earlier, each robot only sees a certain number of closest goals, closest robots and if present closest obstacles. Our results can be seen in Fig. 4.2. The policies in Fig. 4.2(Left) and Fig. 4.2 (Center) are produced by transferring policies trained to utilize information from 3-hop neighbors. In Fig. 4.2 the policies are transferred after being trained to utilize information from 5-hop neighbors. Consider the formation shown in Fig 4.2 (Left). Here each robot receives information about 3 of its nearest goals and these nearest goals overlap with its neighbors. Further, since the goals are very far away and robots are initialized close to each other, a robot and its neighbor receives almost identical information. In such a scenario the robots must communicate with each other and ensure that they each pick a control

action such that they do not collide into each other and at the end of their trajectories, all goals must be covered. The GPG filters learn this local coordination and can be extended to every robot in the swarm. When training with obstacles, we extend the system state in Eqn. 4.1 to include the positions of the obstacles $\mathbf{O}$ i.e $\mathcal{S}(t) = \{\mathbf{R}(t), \mathbf{G}, \mathbf{O}\}$ and extend the state of the robots in Eqn. 4.5 to observe M nearest obstacles. The cost itself is unchanged, and as such all results derived in this paper still hold with the inclusion of the state of the obstacles. As before we train with the GNN filters with a small number of robots and extend the filters to a larger number. We observe in Fig 4.2 (Right) and attached video with this paper that our solution is capable of covering goals even when obstacles are present. We also study the effects of the choice of K and M on the performance of the learned models. We observe that as the number of hops, i.e K each robot communicates with, the performance measured by looking at the success percentage (number of times all goals are covered without collisions during 100 runs) during inference increases. However, this increases only to a certain point after which there is a steep drop off in performance. This can be explained to the increase in input dimensionality at each node, thus making the problem more challenging. This result can be seen in Fig. 4.3 (Left). A similar effect can be seen when the value of M, i.e direct neighbors and goals that each robot senses. The increase in information only improves the learning performance to a certain point after which there is a drop off in performance as seen in Fig. 4.3(Right).

4.4.2 Comparison to Centralized Model Based Methods

To further quantify the performance of GPG, we compare with a model based approach that uses centralized information, called concurrent assignment and planning (CAPT), proposed in [19]. When used in an obstacle free environment, it guarantees collision free trajectories. We direct the reader to [19] for more details about the CAPT algorithm. We set

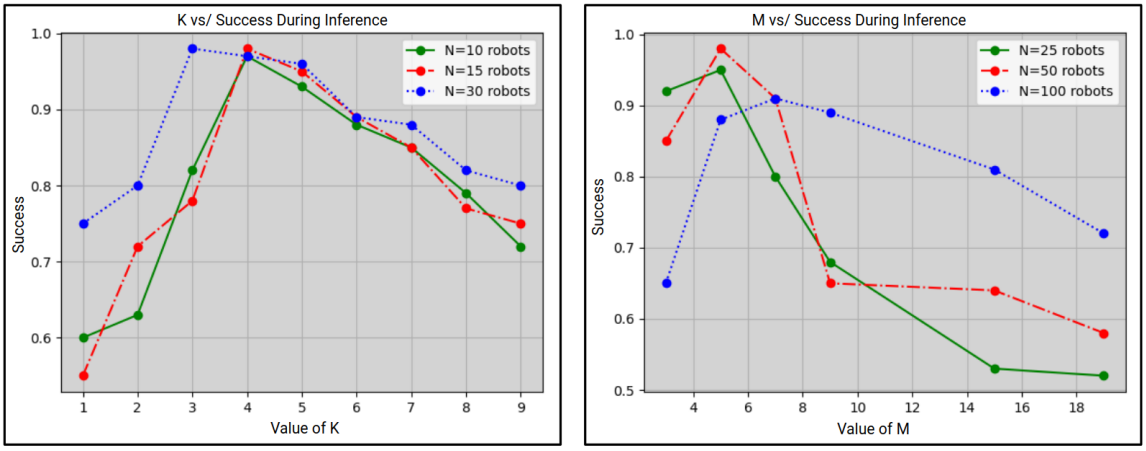


Figure 4.3: Success v/s choice of K and M. (L) During inference, we analyze effects of choices for K and M.

up three different formations F1, F2 and F3 similar to that in Fig 4.2 (Center). On average the goals in F3 are further away from the starting positions of the robots than those in F2 and those in F2 are further away from goals in F1. In this work, we treat CAPT as the oracle and look to compare how well GPG performs when compared to this oracle. We use time to goals as a metric to evaluate GPG against this centralized oracle. Our results can be seen in Fig. 4.4 The key takeaway from this experiment is that decentralized inference using GPG, performs always within an ϵ margin (approximately 12-15 seconds) of the optimal solution and this margin remains more or less constant even if the goals are further away and if the number of robots are increased. However, GPG outshines CAPT when the planning times for both are compared. CAPT employs the hungarian algorithm to solve the task assignment and as a result has a time complexity of $\mathcal{O}(n^3)$ whereas GPG only requires a feedforward pass during inference. The planning times for both methods can be seen Fig. 4.5. It is important to note that in the original CAPT method, the optimal performance guarantees zero collisions. While the solution proposed in this paper is a learned solution with the advantage of being decentralized, we cannot guarantee collision free trajectories. Nevertheless, we analyze the number of collisions between robots. This performance can

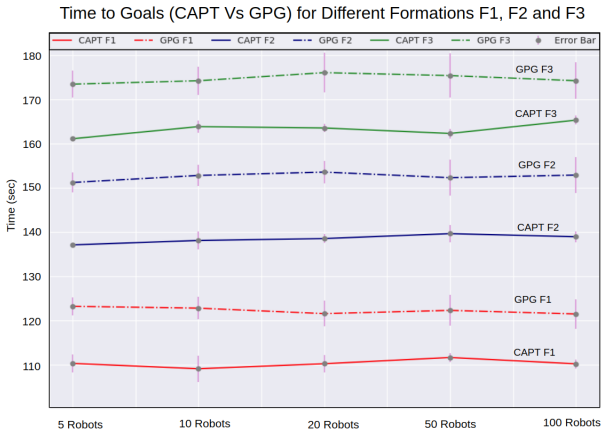


Figure 4.4: Time to Goals Time taken by CAPT to cover all goals v/s time taken by GPG to cover all goals when robots follow velocity controls..

be seen in Table . We argue that while any learned method cannot guarantee collision free trajectories, it has been shown in the Chapter 2 that simple model based *backup* policies can be used in conjunction with learned policies to empirically guarantee collision free trajectories. Similar to the method adopted in Chapter 2, we use velocity obstacles [28] as a backup policy that intervenes every time a collision between two robots executing GPG policies is imminent. We call this method **GPG+VO**. It is important to note that in order to prevent GPG+VO from producing degenerate solutions at the cost of collision free trajectories, we add a penalty term to the cost function every time VO is called during training.

We analyze these results in Table 4.1 for 30 robots over 3 line to circle formations similar to the one seen in Fig 4.2 (Right). The formations R1 < R2 < R3 differ from each other in the size of the radius of the circle on which the goals are distributed. As in Fig 4.4 (Left) we observe that the difference between CAPT and GPG is the same even with different number of formations. In terms of number of collisions when the robots have more open space to operate, the number of collisions are almost reduced to zero with just GPG. When the room to operate is small as in R1 the GPG based policy tends to produce a higher

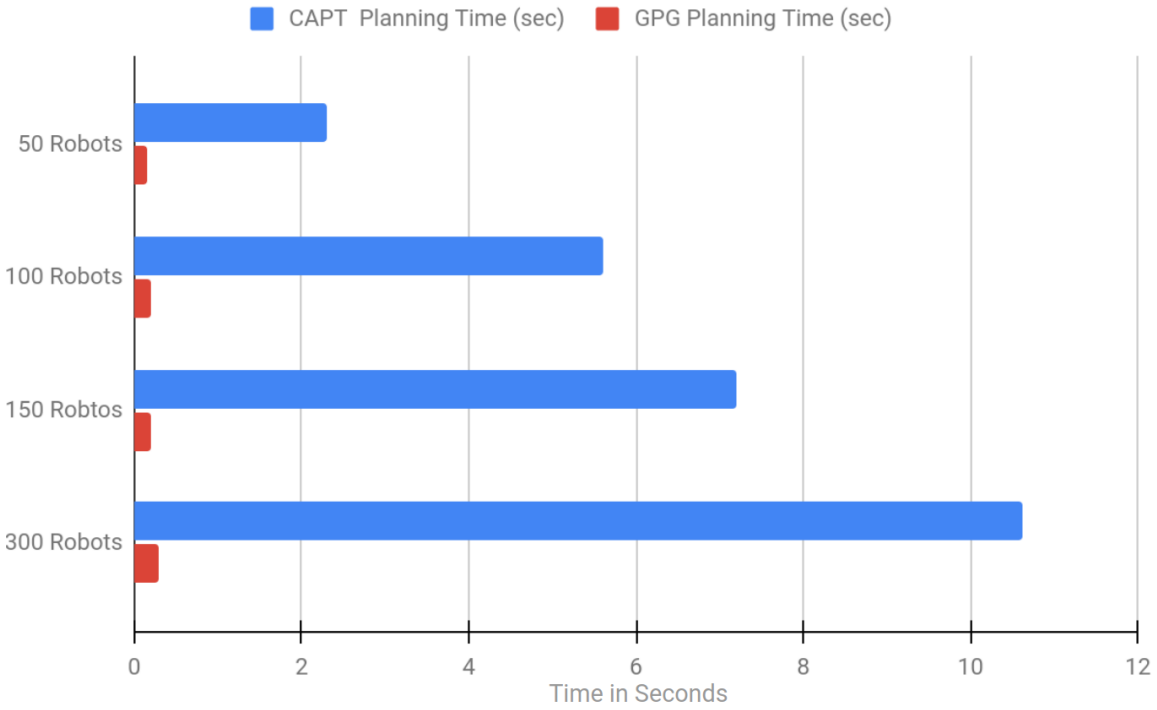


Figure 4.5: Planning Times. CAPT v/s GPG Planning Times.

number of collisions. The backup model based policy in GPG+VO is able to remove the number of collisions but results in an increase in time to cover goals. Using the backup model based velocity obstacles, increases the time required by an almost negligible amount when the robots have significant room to operate such as in R3 and at the same time offers a decentralized solution. Hence, we conclude that decentralized GPG is close in performance to the provably optimal centralized solution.

4.4.3 High Order Dynamics

In the previous experiments, we modeled the robots as ideal point mass systems to study the feasibility of GNNs for the unlabeled motion planning problem. In this experiment we

	CAPT		GPG		GPG+VO	
	T(sec)	C	T(sec)	C	T(sec)	C
R1	68.27	0	85.31	11.4	94.9	0
R2	83.4	0	103.56	8.1	112.62	0
R3	98.16	0	122.15	2.3	124.15	0

Table 4.1: CAPT vs GPG vs GPG+VO. Total time (T(sec)) and total number of collisions (C) during inference for 30 robots over 3 line to circle formations R1, R2 and R3 similar to the one seen in Fig 4.2 (Right). R1 < R2 < R3 differ from each other in the size of the radius of the circle on which the goals are distributed. Averaged over 20 runs.

test our learned policies in the AirSim simulator which allows us to test our policies in the presence of higher order dynamics, slower control rates and latency in observations thus mimicking a real world robot swarm more realistically. In such a setting it is even more imperative to be able to train with a smaller number of robots and be able to transfer to a larger number without any additional training samples due to the increased computational complexity associated with training a large number of robots directly in the simulator. In this setting, the outputs from the policies are interpreted as accelerations and are converted to desired roll and pitch commands which are fed into the simulator. Further, since our proposed framework is model free, it can still be trained directly as before with only a sparse reward. Empirically, in Fig 4.6 and the attached video we observe our The action $\mathbf{S}_{nt}$ chosen by the robot gives the change in velocities. We execute the same paradigm as before. The unlabeled motion planning is trained for a small number of robots. Then, during inference time the filters learned for the small swarm are used across all the robots. A snapshot of the performance of our algorithm can be seen in Fig 4.6.

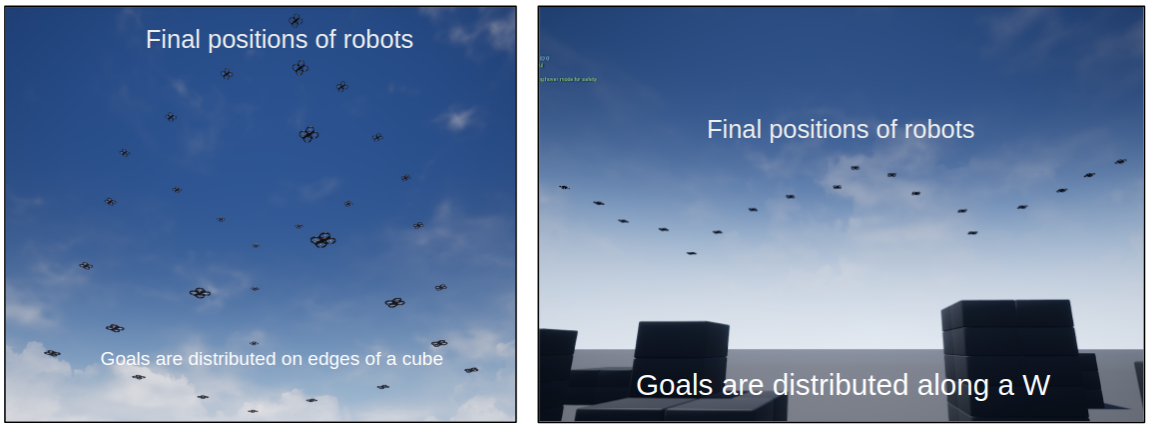


Figure 4.6: Large Scale Unlabelled Motion Planning in AirSim using GPG. Control to the training algorithm is handed after robots are at a certain altitude. (L) During inference, we use the trained filters to cover goals spread on the edges of a cube. (R) Goals to be covered are spread along a W.

4.5 DISCUSSION AND GUIDING IDEAS

So far, we have used the GPG method to achieve scalable decentralized solutions for the full unlabelled motion planning problem. We show that GPG trained policies can be transferred over to a larger number of robots and the solution computed is close to the solution computed by a centralized *oracle*. This closes one of the threads of inquiry that this thesis opened in Chapter 2 namely, *can we learn a model free solution to the unlabeled motion planning problem that improves over model based solutions but can also match up to model based solutions when scaling to a larger team of robots.* However, our solution(s) are far from complete and make many assumptions. First and foremost, all the work in Chapter 2,3,4 assumes perfect information about the state of the local world. Further, we also assume perfect communication rates even among local robots. Lastly, our solutions are purely reactive as they have no notion of history, In the next chapter, we shall look to tackle some of these assumptions and look to build perception-action-communication loops for teams

of robots operating in environments where simple reactive policies can lead to catastrophic collisions and failures.

5 LEARNING DECENTRALIZED PERCEPTION ACTION COMMUNICATION LOOPS

In the recent past, deep learning has played an immense role in advancing the state of the art in robotics. Several papers have shown convolutional neural networks (CNNs) to be immensely helpful when attempting to learn control policies directly from images for a variety of tasks. For example, CNNs have been successfully used to learn policies for robot arms [12], race drones [56], navigate robots in unknown environments [57]. While most of these works deal with single robot scenarios, in our earlier Chapter 2-4 we demonstrated how one might go about applying deep learning to learn scalable decentralized policies for teams of robots albeit with simpler sensors or perfect state information. In this chapter, we look to build upon the limitations of Graph Policy Gradients by attempting to learn scalable decentralized policies for teams of robots operating in constrained environments where each robot relies only on its own visual and inertial sensors and information from its local neighbors. An example of this could be a team of robots traversing a forest or a team of robots looking to navigate an obstacle course (5.1).

Consider the problem of learning to fly a team of quadrotors through a constrained environment such as a forest or a series of checkpoints such as the one seen in Fig 5.1 where each robot is only equipped with a front facing camera and an inertial measurement unit (IMU). The robots are tasked with reaching a given goal waypoint. A somewhat plausible first attempt might involve collecting ground truth data using an expert trajectory and training a CNN policy for each robot. However, such an approach would fail to generalize

Figure 5.1: **Learning Decentralized Perception Action Communication Loops for Robot Teams**. Relying only on onboard visual and inertial sensors, robots compute decentralized control policies to fly through the constrained environment.

to a new obstacle course. Such a CNN policy would be completely reactive and possibly produce degenerate solutions since at each timestep, the robot only senses a partial view of the world and has no notion of where its neighbors are. If we were to assume robots can also communicate with each other, the complexity of the problem increases since we must now compute the correct subset of neighbors each robot must communicate with in order to execute meaningful control actions. We would also need to take into account that each robot only has a noisy estimate of its own position since IMUs/Visual Inertial Odometry (VIO) systems are perceptible to drift over time. Alternatively, training the CNN policy for each robot with reinforcement learning (RL) might negate the need for generating expert trajectories and somewhat compensate for the noisy state information, but faces other challenges such as the need for a large number of samples to train the policy, designing a local reward function that is consistent with desired global behavior and challenges in training with a large number of robots. We hypothesize that the problem of learning scalable perception action loops for teams of robots can be solved by decomposing it into two parts; 1) learning a robust perception model and 2) learning decentralized policies for the team of robots by exploiting local structure among the robots.

Perception System In our proposed solution the perception model uses an off the shelf convolutional neural network (CNN) to identify waypoints that each robot must navigate to. A dataset of camera images and corresponding waypoints is computed by running a trajectory planner for a single robot in a known environment during training time on which the CNN is trained. It is also important to note that we only collect ground truth data for one robot and train a single perception model that we aim to use across all robots during inference.

Collaborative Decentralized Policies In order to learn decentralized control policies for multiple robots using a central reward function, we propose using multi-agent RL (MARL) to train our policy network. However, in order for the system to be scalable to a large number of robots, we look to exploit the inherent graph structure among the robots. In Chapters 3 and 4 we demonstrated that graph neural networks (GNNs) [33, 40] can be a good candidate to parameterize policies for robots as opposed to the fully connected networks generally used in MARL. Define a graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ where $\mathbf{V}$ is the set of nodes representing the robots and $\mathbf{E}$ is the set of edges defining relationships between them. Similar to before, we define edges between robots based on proximity relationships. This graph acts as a support for the data vector $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top$ where $\mathbf{x}_n$ is the output from the perception model combined with an estimated state representation of robot n. The output of the GNN is interpreted as $\Pi = [\pi_1, \ldots, \pi_N]$, where $\pi_1, \ldots, \pi_N$ are independent control policies for the robots. However, due to the dynamic nature of the world where the robots are evolving in space and time, the underlying graph keeps and robot n's neighbors keep evolving. In order to produce robust control policies that work with partial information, we propose a new *memory GNN* architecture that incorporates a learned memory component at each node thus, maintaining a notion of history over time. We look to demonstrate the efficacy of our proposed architecture in two scenarios, an environment cluttered with trees where the robots are tasked with reaching goals at some

distance without colliding into each other or the trees and an environment populated with checkpoints that the robots must traverse in the right order while avoiding collisions.

5.1 METHODOLOGY

5.1.1 Preliminaries

We pose the problem of learning distributed perception action communication loops for a team of robots as a policy learning problem in a collaborative Markov team [24]. The team is composed of $\mathbf{N}$ robots generically indexed by n which at any given point in time t occupy a position or pose $p_{nt} \in \mathbb{R}^3$ in configuration space and must choose an action $\mathbf{a}_{nt} \in \mathcal{A}$ in action space. For the sake of notation, we drop the time index unless otherwise specified.

At each time step robot n has access to an image I_n from a front facing camera. To simulate communication, we also assume that robot n also has a fixed communication radius ϵ and can communicate with at most $k (k << \mathbf{N})$ nearest robots inside ϵ. Let the information from these k nearest robot be represented as $\boldsymbol{\nu}_n = \{\nu_1, \ldots, \nu_k\}$ where ν_k is some information communicated by robot k to robot n. It is important to note that this set of k nearest neighbors is *not fixed* since the robots are evolving in space and time and the set of k robots closest to n can change from one timestep to another. Thus the state of robot n at any given time can be given as $\mathbf{x}_n = [I_n, \boldsymbol{\nu}_n]$. Given access to this robot state $\mathbf{x}_n$, robot n chooses an e-dimensional action $\mathbf{a}_n \in \mathbb{R}^e$ that controls the pose of the robot according to some unknown underlying dynamical model. To best capture the stochastic nature of the dynamics model, the robot samples *continuous* actions $a_n := \pi_n(\mathbf{x}_n | \mathbf{a}_n)$ from a policy π_n. Collectively, for the robot swarm we denote $\Pi = [\pi_1 \ldots \pi_N]$. Additionally,

the team and environment are assumed to be Markovian. This means that if one were to collect the robot configurations in the vector $\mathbf{x}_t := [\mathbf{x}_{1t}, \ldots, \mathbf{x}_{Nt}]^\top$ and actions in the vector $\mathbf{a}_t := [\mathbf{a}_{1t}, \ldots, \mathbf{a}_{Nt}]$ then, the evolution of the system is completely determined by some conditional transition probability $\mathcal{T}(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t)$. Each robot is tasked with navigating to a goal in a collision free manner, i.e robot n must reach goal $g_n \in \mathbb{R}^3$ in minimum time and avoid collisions with other robots and other entities present in the environment. As robots operate in the environment, they collect a *centralized* global reward r_t. The objective for the whole team is then to compute actions for all $\mathbf{a}_t := \Pi(\mathbf{a}_t|\mathbf{x}_t)$ such that the expected sum of rewards over some time horizon T is maximized:

$$\sum_{n=1}^{N} \max_{\theta} \mathbb{E}_{\Pi}\left[\sum_{t}^{T} r_t\right] \tag{5.1}$$

where θ are the parameters of Π. In the next section, we introduce our modular approach we call Graph Memory Policies (Fig 6.3) to learn Π that which consists of learning an offline perception system and learning a distributed control policy scheme with limited communication with nearby robots.

5.1.2 Perception System

Robot n is equipped with a front facing camera that at each timestep returns a 300×200 image I_n. The perception subsytem takes in as input this camera image I_n and predicts a desired direction that the robot must navigate to. The output of the perception system is a two-dimensional vector that encodes the direction to the next waypoint in normalized image co-ordinates. Estimating image coordinates helps eliminate the issues with drift that creeps in when using global co-ordinates. This approach has been shown to be useful for drone racing [58] with deep learning.

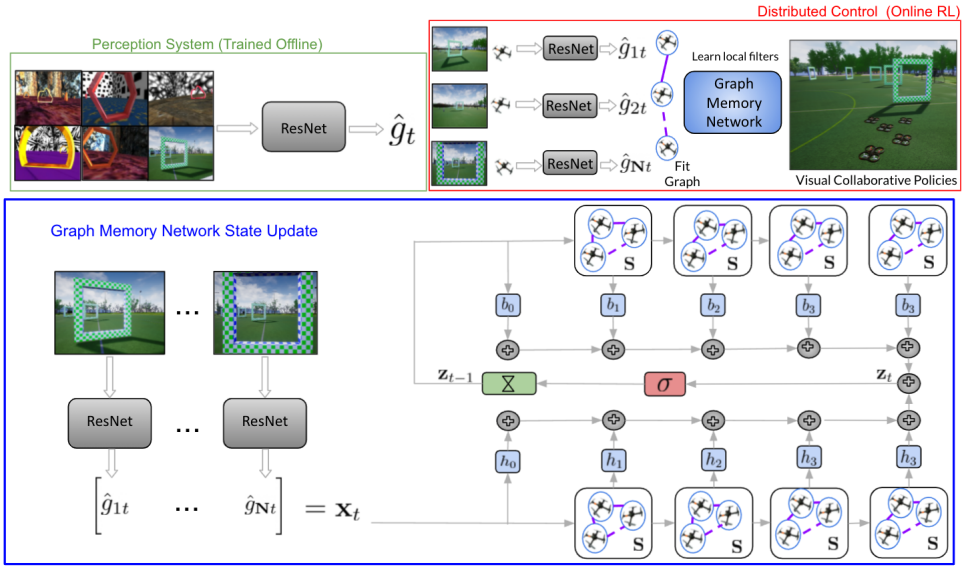


Figure 5.2: **Modular Approach for Decentralized Perception Action Communications Loops for Robot Teams** The proposed decentralized solution to co-ordinate a team of robots in a constrained environment with limited communication consists of two subsystems. **Top L** The perception subsystem is trained offline for a single robot to predict waypoints that the robot must navigate to. **Top R** The control subsystem uses a graph memory neural network that takes in the prediction from the vision subsystem and communicates with nearby robots to collaboratively fly through the obstacle course. **Bottom** State computation in a single layer of a graph memory network with $K = 4$. The blue blocks represent linear weights, red blocks represent non-linearity and the green block represents a time shift. We stack multiple such layers and the output of the final layer is Π.

The perception subsystem here uses a modified version of [59] and can be seen in 5.3. The input to the perception system is a 32×32 greyscale image and consists of two residual blocks. The output of the perception system $\hat{g}_t$ at time t is a two dimensional vector that represents a goal that the robot should navigate towards. The perception system is trained by minimizing the L2 error between the predicted $\hat{g}_t$ and true label $\tilde{g}_t$. The network is trained using Adam and a learning rate of $1e-3$.

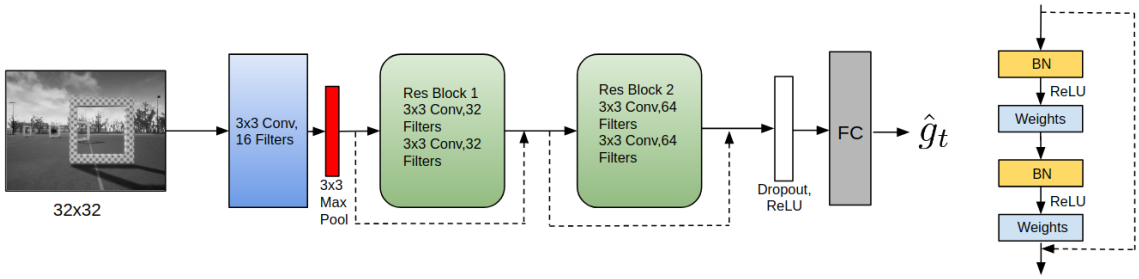


Figure 5.3: **Perception System to Predict Waypoints. Left** Full perception system to predict waypoints. **Right** Individual Residual Blocks.

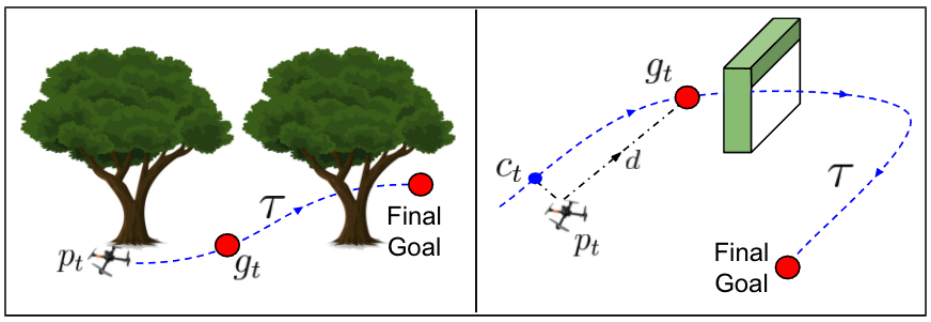


Figure 5.4: **Data Collection.**

5.1.3 Dataset Generation

In order to train the perception system we first need to collect a dataset D. To generate a training dataset mapping images to intermediate goal positions, a global trajectory τ that avoids obstacles and reaches the goal is computed by leveraging the work of [60]. This trajectory planner takes in as input the start state of the robot, the final goal state, and intermediate waypoints, see Fig 5.4 and produces a velocity profile/trajectory τ for the robot to follow. The intermediate waypoints are selected randomly in free space or at the center of a checkpoint. A receding horizon controller tracks this reference trajectory for a single quadrotor. At any given time, the quadrotor's pose is given by p_t. Let $c_t \in \mathbb{R}^3$ be the pose closest to the quadrotor lying on the global trajectory τ. At every time t a desired goal

position g_t which lies at distance d from p_t, lies on the trajectory τ and is in the forward direction with reference to c_t is computed. Finally, the desired goal position g_t is projected into the image plane of the camera mounted on the front of the quadrotor to get the goal co-ordinates in the image plane, $\tilde{g}_t$ which along with the camera image I_t is recorded in the dataset; $D := \{I_t, \tilde{g}_t\}$. In order to generalize to different obstacle courses and be robust to other noise such as partial occlusions from other robots during inference, we leverage domain randomization [61] and varying factors such as illumination, viewpoint, background textures, etc are randomized and added to the dataset D. Let the perception system be denoted by $\mathcal{F}(I_t) : I_t \rightarrow \tilde{g}_t$ produce a predicted goal coordinate $\hat{g}_t \in [-1, 1]^2$, given an image I_t. It is trained over the dataset D by minimizing $\|\hat{g}_t - \tilde{g}_t\|_2$.

For the perception system, we need to train a system that takes in an image of the environment and a representation of the goal and outputs a waypoint that drives the robot towards the goal and avoids collisions with trees at the same time. Let the robots start position given by p_{t_0} and the final goal pose that the robot needs to get to be given by p_{t_T}. We pick a collection of intermediate points $[p_{t_1}, \ldots, p_{t_{T-1}}]$ that lie in free space. By leveraging the work of [60] we generate a global trajectory τ that starts at p_{t_0} passes through these intermediate points and terminates at p_{t_T}. Computing this trajectory is a centralized process, since we need to know the dimensions of the map/locations of the trees to pick intermediate goal points that lie in free space. Once τ is generated, we use a receding horizon controller to track τ. At every time instant, if the robots position is p_t, the closest point c_t lying on τ is computed. Then, a desired goal position g_t for time t is generated by computing a point that is at a distance d from p_t and in the forward direction with reference to c_t. Since, we care about predicting things in image co-ordinates, we backproject this desired goal g_t to the image plane and call it $\tilde{g}_t$. This $\tilde{g}_t$ serves as the label for the input image I_t at time t.

Dataset Generation for Environment with Gates

In the case of the environment with gates, we have a sequence of gates we would like the robots to traverse in a specific sequence, i.e populated by a series of M of *checkpoints* or *gates* $\mathbf{G} = \{\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_M\}$ such that the gates must be traversed in order, i.e $\mathbf{G}_1 \rightarrow , \mathbf{G}_2, \rightarrow \ldots \mathbf{G}_M$ by all N robots. Each gate $\mathbf{G}_m$ is defined as a square perpendicular to the z-axis and is defined by a tuple $\langle \overrightarrow{n_m}, \overrightarrow{c_m}, h_m \rangle$ where $\overrightarrow{n_m}$ is the normal unit vector to the gate plane, $\overrightarrow{c_m}$ is the vector from the origin of the world to the center of the gate plane and h_m is the scalar height of the gate. A gate is said to be traversed by robot n with pose $p_n(t)$ at some time t if the following conditions are met:

$$(\overrightarrow{p_n}(t) - \overrightarrow{c_m}).\overrightarrow{n}_m = 0 \ \wedge \ \|(\overrightarrow{p_n}(t) - \overrightarrow{c_m})\|_1 \leqslant h_m \tag{5.2}$$

where $\|.\|_1$ is the L-1 norm of a vector. More formally, for each robot-gate pair we can define a function

$$F_n^m(t) = \mathbb{I}\left[(\overrightarrow{p_n}(t) - \overrightarrow{c_m}).\overrightarrow{n}_m = 0 \wedge \|(\overrightarrow{p_n}(t) - \overrightarrow{c_m})\|_1 \leqslant h_m\right] \tag{5.3}$$

where $F_n^m(t) = 1$ when the two conditions for traversing a gate are satisfied. In this setting, in order to generate τ, we set the intermediate goals to lie at the center of the gates. As before, we generate a global trajectory that takes in as input the start position of the robots, these intermediate goal points and the final goal point. A receding horizon controller tracks this trajectory and we record intermediate image/desired goal position tuples as before.

Domain Randomization

While generating the dataset D, we only consider the existence of a single robot. This does not account for occlusions from other robots or lend adaptability to different courses. Some examples of the different environments we consider can be seen below. For the forest environment, we generate instances of an environment populated with trees in a physics based simulator. Our choice of simulator here is [50] and we generate different environment instances by varying the density of the trees and varying factors such as illumination/time of day etc. For the environment with gates, we leverage the work in drone racing [58] and add to the dataset by using environments from AirSim.

While the perception system outputs a series of waypoints given camera inputs that drive the robot to the goal position, it does not take into account the existence of other robots or constraints induced from team objectives. If each robot is given information about the positions of all other robots, in theory it would be possible to design a collaborative control policy for each robot. However, such a centralized scheme can become infeasible due to communication constraints and the increase in dimensionality as the number of robots increase. As such, we motivate the need for decentralized control policies for each of the robots that relies only on local information from its own sensors and communication from nearby neighbors to achieve a high level team wide objective. In the recent past, graph neural networks have been shown to be a viable solution for designing local controllers

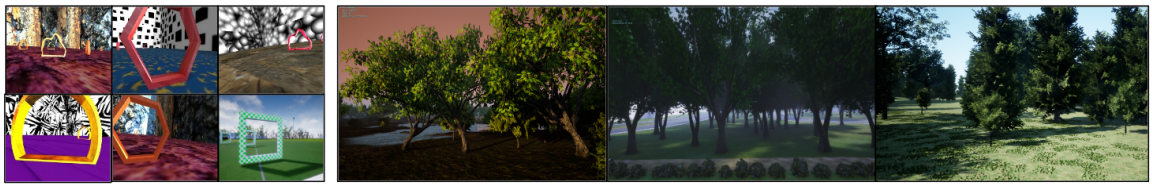


Figure 5.5: **DataSet Collection. Left** Dataset Instances for Collaborative Flight through Gates. **Right** Dataset Instances for Collaborative Flight through Cluttered Environments.

for a large team of robots that maximize a high level team wide cost function [62, 63]. A drawback of these works is that they assume perfect state information and noiseless communication among the robots. In this work, the controller operates on the outputs of the perception system which are inherently noisy and we also look to operate the teams of robots over longer horizons of time which necessitates the need for robust control policies that have a notion of history. In the next sections, we introduce graph memory neural networks and propose Graph Memory Policies (GMP) to learn perception-action-communication loops for robots.

5.1.4 Graph Memory Networks and Graph Memory Policies

Consider a graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ described by a set of $\mathbf{N}$ nodes denoted $\mathbf{V}$, and a set of edges denoted $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$. This graph is considered as the support for a data signal $\mathbf{x}_t = [x_{1t}, \ldots, x_{Nt}]^\top$ where the value x_{nt} is assigned to node n at time t. The relation between $\mathbf{x}_t$ and $\mathcal{G}$ is given by a a matrix $\mathbf{S}$ called the graph shift operator. The elements of $\mathbf{S}$ given as s_{ij} respect the sparsity of the graph, i.e $s_{ij} = 0$, for all $i \neq j$ and $(i, j) \notin \mathbf{E}$.

Valid examples for $\mathbf{S}$ are the adjacency matrix, the graph laplacian, and the random walk matrix. $\mathbf{S}$ defines a map $\mathbf{y}_t = \mathbf{S}\mathbf{x}_t$ between graph signals that represents local exchange of information between a node and its one-hop neighbors. More concretely, if the set of neighbors of node n is given by $\mathfrak{B}_n$ then $y_{nt} = [\mathbf{S}\mathbf{x}_t]_n = \sum_{j=n, j \in \mathfrak{B}_n} s_{nj} x_{nt}$. This operation performs an aggregation of data at node n from its neighbors that are one-hop away at time t. The aggregation of data at all nodes in the graph is denoted $\mathbf{y}_t = [y_{1t}, \ldots, y_{Nt}]$. By repeating this operation, one can access information from nodes located further away. For

example, $\mathbf{y}_t^k = \mathbf{S}^k\mathbf{x}_t = \mathbf{S}(\mathbf{S}^{k-1}\mathbf{x}_t)$ aggregates information from its k-hop neighbors. Now one can define the spectral K-localized graph convolution at time t as :

$$\mathbf{z}_t = \sigma \sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{x}_t = \sigma \mathbf{H}(\mathbf{S})\mathbf{x_t} \tag{5.4}$$

where $\mathbf{H}(\mathbf{S}) = \sum_{k=0}^{K} h_k \mathbf{S}^k$ is a linear shift invariant graph filter [45] with coefficients h_k, K is a user set parameter that defines how many hops we would like to aggregate information over and similar to CNNs the output of the GCN is fed into a pointwise non-linear function σ. To introduce memory at each node, we leverage the fact that the output of a graph neural network is also a graph in itself. Thus, we assume that the state $\mathbf{z}_t$ is in itself a hidden nodal state. By defining another linear shift operator, $\mathbf{B}(\mathbf{S}) = \sum_{k=0}^{K} b_k \mathbf{S}^k$ with coefficients b_k, we can write the graph memory neural network (GMN) output as

$$\mathbf{z}_t = \sigma(\mathbf{H}(\mathbf{S})\mathbf{x_t} + \mathbf{B}(\mathbf{s})\mathbf{z_{t-1}}) \tag{5.5}$$

This representation is similar to a recurrent neural network where a hidden state is updated over a sequence while being provided with an input sequence and the output at any given time t is computed by a linear combination of the hidden state at time t and the sequence input at time t. A visual representation of a graph memory neural network can be seen in Fig 6.3, bottom. In order to learn a Π that maximizes the reward in Eqn. 6.2, we propose parameterizing Π with a Graph Memory Neural Network (GMN). More formally, let the state of robot n at time be given as $\mathbf{x}_n t = \mathcal{F}(I_{nt})$, i.e the output from the perception system. The robots can be represented as a graph $\mathcal{G}$ with $\mathbf{N}$ nodes where each robot represents a node in the graph and the edges are based on proximity relationships. This graph acts as the support for the data vector $\mathbf{x}_t = [\mathbf{x}_{1t}, \ldots, \mathbf{x}_{Nt}]$. To compute the policies, a GMN

architecture with L layers is initialized. At each layer according to Eq.6.7, the output is given as:

$$\mathbf{z}_t^{l+1} = \sigma\big(\mathbf{H}(\mathbf{S})\mathbf{z}_t^{l-1} + \mathbf{B}(s)\mathbf{z}_{t-1}^{l}\big) \tag{5.6}$$

where σ is a pointwise non-linear function, $\mathbf{z}_t^0 = \mathbf{x}_t$ and $\mathbf{z}^L = \Pi = [\pi_1, \ldots, \pi_N]$. In practice, the final layer outputs are parameters of Gaussian distributions from which actions are sampled. Intuitively at every node, the GMN architecture aggregates information and uses this information to compute policies. While rolling out a trajectory at each timestep t each robot receives the same centralized reward r_t (defined in Sec 5.2 and Sec 5.3) and attempts to learn a policy that best optimizes this reward. It is assumed that the policies for the robots are independent. The overall objective function function for all the robots as given in Eq.6.2

$$J = \sum_{n=1}^{N} \max_{\theta} \mathbb{E}_{\Pi}\Big[\sum_{t}^{T} r_t\Big] \tag{5.7}$$

where θ now represents the filter weights of the GMN for Π. Consider a trajectory $\tau = (\mathbf{x_0}, \mathbf{a_0}, \ldots, \mathbf{x_T}, \mathbf{a_T})$. Since the reward along a trajectory is the same for all robots and all robot policies are assumed independent, using direct differentiation the policy gradient is given as :

$$\begin{aligned}\nabla_\theta J = \mathbb{E}_{\tau\sim(\pi_1,\ldots,\pi_N)}\Bigg[\Big(\sum_{t=1}^{T} \nabla_\theta \log[\pi_1(\mathbf{a}_{1t}|\mathbf{x}_{1t}) \\ \ldots \pi_N(\mathbf{a}_{Nt}|\mathbf{x}_{Nt})]\Big)\Big(\sum_{t=1}^{T} r_t\Big)\Bigg]\end{aligned} \tag{5.8}$$

The weights θ of the GMN, are then updated using any variant of stochastic gradient descent. We call this algorithm Graph Memory Policies (GMP). In the next section we demonstrate how GMP is able to learn meaningful policies in a variety of settings and scenarios.

5.2 COLLABORATIVE FLIGHT THROUGH A CLUTTERED ENVIRONMENT

We test the ability of GMP to maneuver a team or robots through a cluttered outdoor environment. Our choice of environment here is a simulated environment consisting of trees that represent obstacles. The robot swarm is initialized in an open space with the camera plane facing the direction of the goal heading.

Each robot is also assigned a goal at some distance. In this task, the state space of each robot is appended to include the relative distance between robot's current position and goal position; i.e $\mathbf{x}_{nt} = [I_{nt}, p_{nt} - G_n]$ where G_n is the final goal assigned to robot n. The graph $\mathcal{G}$ is constructed by considering each robot as a node and edges are added for robots that are within ϵ distance of each other with a maximum degree of any node is capped at 3. The perception subsystem $\mathcal{F}$ takes in as input the image I_{nt} and predicts an intermediate waypoint $\hat{g}_{nt}$. The state space of robot n that is fed into the decentralized control system is then represented as $\mathbf{x}_{nt} = [\hat{g}_{nt}, p_{nt} - G_n]$. Our choice of state space representation is to conserve the permutation invariance property as this can be useful when training for a large number of robots with a graph neural network [64, 63]. To investigate the emergence of interesting collaborative behaviors, we also enforce a region $\mathcal{C}_t \in \mathbb{R}^3$, a region of fixed size around the robots, in which the robots must be contained in at all times. To capture this behavior, we design the following high level reward function

$$r_t = \sum_{i=1}^{N} \|p_{it} - G_i\| + \lambda_1 \sum_{i=1}^{N} 1_{\mathcal{C}_t}(p_{it}) - \lambda_2 \sum_{i=1}^{N} E(p_{it}) \tag{5.9}$$

where $\sum_{i=1}^{N} 1_{\mathcal{C}}(p_{it})$ is the indicator function that returns 1 if robot n is contained in region $\mathcal{C}$ and 0 otherwise, $E(p_{nt})$ returns a scalar value if robot n is in collision and λ_1 and λ_2 are

positive hyper parameters. The same reward r_t is given to all robots. We investigate the results of GMP by comparing with a centralized baseline that has access to full information about the state of all entities in the environment and the positions of all robots. The centralized planner is an online planner that recomputes trajectories for all robots at every time step. It does so by considering all other robots as an obstacle. While this centralized baseline is by no means an optimal solution, it provides a good benchmark when the robots are moving slowly enough or the time horizon for replanning is small enough.

	10 Robots	**25 Robots**	**35 Robots**
CNN	8.23	18.81	29.07
GNN	3.95	10.52	16.45
GMP	**1.51**	**4.4**	**9.37**

Table 5.1: Number of collisions per robot for the forest environment. Results averaged over 50 runs.

For collaborative flight through a cluttered environment we reuse environments from AirSim, a high fidelity physic simulator [50]. To experiment with varying levels of tree coverage, we reuse the tree assets from the simulator and spawn them in a fixed area of operation according to a random Poisson process. The robots start in a specific area of the environment but do not have fixed starting positions. Similarly, goals are defined in a specific area but are randomized between runs. The constraint region is defined as a function of the number of the robots. Each robot is treated as a circular disk with radius, R. If there exist $\mathbf{N}$ robots operating, then the dimensions of the constraints box are given as $2R(\mathbf{N}+2) \times 2R(\mathbf{N}+2)$. Since the output of the robots is only in the XY plane, the existence of such a constraint box necessitates team-wide collaboration. The centralized baseline is generated by leveraging the work of [65] which is a search based planner for quadrotors. By treating all the other robots and obstacles in the map as static, the centralized baseline finds a valid trajectory for a time horizon dt. At the end of dt the search based planner

recomputes a new trajectory with the new positions of the robots. We would like to point out that such a method is only a hypothetical exercise for the sake of comparing the learned methodologies with a clairvoyant that has access to all information but it would be almost infeasible to execute on a real team of robots.

The maximum robot velocities are clamped depending on the number of robots present in the configuration; for example, for a team of 5 robots, the max velocity is capped at 10m/s, for a team of 35 robots, the max velocity is clamped at 2.5 m/s (Please refer to the appendix for more experimental details). The size of the constraint region is also increased as we increase the size of the robot team. We set $K = 3$ for all our experiments; i.e robots have access to information from neighbors 3 hops away. For all the runs, the distance between start positions of the robots and goal positions is fixed; the length of the cluttered forest remains the same at a distance of 100 meters. **Results** In the first part, we consider time to clear the forest by GMP against the centralized baseline and vary the density of trees/sq meter for different sizes of robot teams. The time to clear is given by the sum of time taken to compute the plan and time taken to execute. For the GMP method, at inference time, we only need to execute a forward inference through our perception network and our graph memory network. For the centralized planner we need to recompute the plan at each timestep. We observe that the centralized planner performs well on small robot teams but the time taken to clear grows almost exponentially as the number of robots increases (Fig 5.6 L). These results are averaged over 5 runs that are collision free. Next, we compute the amount of time that the robots violate the constraint region space and compare that with the centralized baseline method. We find that GMP trades some efficiency in planning time for performance as the complexity of the system grows (Fig 5.6 C). Lastly, to demonstrate the need for memory over a regular GNN, we compare the time to clear by GMP vs a standard 2 layer GNN using the method introduced in Chapter 3 with $K = 3$ that is trained on the same reward and policy gradient method

as GMP. We observe that GMP produces better results than the GNN solution which has no memory. It is important to note that though the GNN solution does worse than the GMP solution, it does outperform than the centralized baseline method (Fig 5.6 R), thus validating the emergence of local structure in our problem. Lastly, we bring to the readers attention that the proposed methods here are not guaranteed to be collision free as they are all learning based solutions. However, it has been shown in Chapter 2 and 4 that it is easy enough to design a model based backup policy on top of a learning solution to get empirically guaranteed collision free trajectories. Nonetheless, we compare the number of collisions for our robots with the CNN policy (the output of the perception system is directly transformed to control outputs), the GNN policy with no memory and GMP policy in Table 5.1 to show that GMP produces solutions of a higher quality.

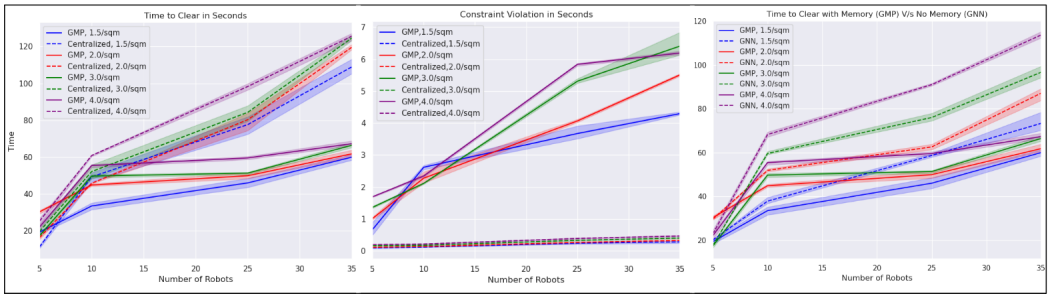


Figure 5.6: Collaborative Flight through a Cluttered Forest. Left Time taken by GMP as compared to time taken by the centralized online planner to clear the forest. Here, time to clear is the sum of planning time and execution time. As the complexity of the environment grows, the online centralized planner has an exponential growth in time to clear. **Center** Total time over all robots for which the area constraints are violated. **Right** Time taken by GMP which has memory v/s a GNN which has no memory. The GNN solution outperforms centralized planner, but produces slower solutions than GMP.

5.3 COLLABORATIVE FLIGHT THROUGH GATES

In the forest environment, each robot was assigned a specific goal to reach. In this experiment, we look to investigate the feasibility of GMP to co-ordinate a team of robots over a more abstract high level team wide objective; co-ordinating distances without giving the robots a specific goal heading. In this section, we look to navigate a series of **M** checkpoints without giving a specific heading to the robots. The checkpoints are arranged similar to a drone racing course as seen in Fig 5.1 (right). The robots are tasked with traversing as many checkpoints as possible in a given time T. Analogous to the setup in Sec 5.2, the image captured by each robot is processed by the perception subsystem to produce an intermediate goal waypoint $\hat{g}_{nt}$. The state space of robot n is $x_{nt} = [\hat{g}_{nt}]$.

	10 Robots	**25 Robots**	**35 Robots**
CNN	4.2	3.3	0.8
GNN	15.6	13.24	10.96
GMP	**21.1**	**18.88**	**16.3**

Table 5.2: Number of gates traversed before collision. Results averaged over 50 runs.

We also enforce the existence of a constraint region $\mathcal{C}_t$ that the robots must operate in. For this task we compute a score function U_t that at any given time t returns the number of gates that have been traversed by all robots from 0 to t. For example, if all robots have crossed the first gate at time t, then $U_{t+1} = 1$. To capture the behavior, the reward function for this task is given as:

$$r_t = U_t + \lambda_1 \sum_{i=1}^{N} 1_{\mathcal{C}_t}(p_{it}) - \lambda_2 \sum_{i=1}^{N} E(p_{it}) \tag{5.10}$$

where as before $\sum_{i=1}^{N} 1_{\mathcal{C}}(p_{it})$ is the indicator function that returns 1 if robot n is contained in region $\mathcal{C}$ and 0 otherwise, $E(p_{nt})$ returns a scalar value if robot n is in collision and λ_1

and λ_2 are positive hyper parameters. Just as before the maximum robot velocities are capped differently based on the number of robots and we set $K = 3$ for all experiments.

Results Three courses are setup namely L1, L2 and L3. These primarily differ from each other in terms of length in that L1 > L2 > L3. Additionally there also exist some variations in offsets/distances between one set of gates to the other such that L1 is the easiest course to traverse and L3 is the hardest course to traverse. (We refer the reader to the appendix for more details). As before, we compare GMP with our centralized planner that runs an online trajectory replanning at every timestep, by treating all other obstacles as fixed. In this experiment, there exist significantly lesser number of obstacles that a robot can collide with and the centralized planner produces results much better than those in the cluttered forest . However, GMP is still able to outperform the centralized planner especially as the number of robots grow due to the increased planning time required for the centralized planner (see Fig 5.7 left). Similar to the experiment in the cluttered forest, the GMP method trades some performance for optimality and violates the region constraints more often than the centralized planner (see Fig 5.7 left). The last comparison exists between GMP and a GNN solution with no memory. In this setting, we observe that the GMP gains over the GNN solution while modest, exist nonetheless. As before, since all methods are learning based methods and guarantee no collision free trajectories, we also analyze the number of gates each method is able to traverse before any one of the robots collide in Table 5.2. In this experiment, we setup a circular track and let the robots execute the policies until there is a collision. We observe GMP is able to traverse a larger number of gates as compared to the other methods before there is a collision.

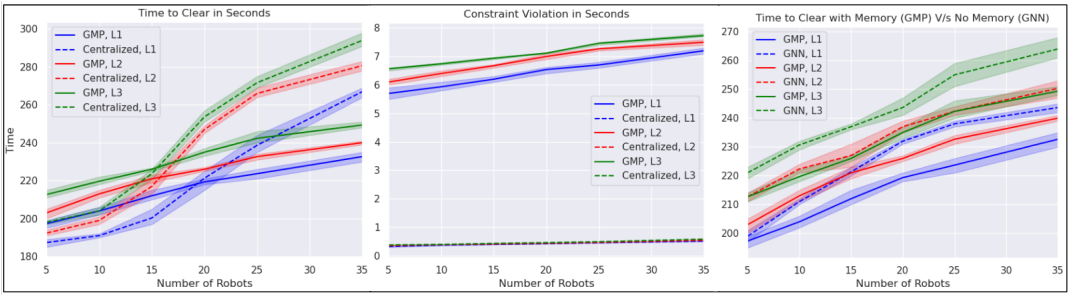


Figure 5.7: Collaborative Flight through a Series of Gates. Left Time taken by GMP as compared to time taken by the centralized online planner to clear the forest. **Center** Total time robots violate area constraints. **Right** Time taken by GMP which has memory v/s a GNN which has no memory.

5.3.1 Imperfect Communication

The limiting factor so far, is the assumption of instantaneous and perfect communication at all times with the neighboring robots. In order to investigate the limits of this assumption, we looks to vary the velocities of the robots and use that as a proxy for communication rates. Our results can be found in Fig 5.8 (Left). We observe that the centralized baseline performs well when the robots are moving slowly. However, when the robots are moving faster, the performance of the system degrades. The replanning time horizon for the centralized baseline does not change as the velocity of the robots is increased. While the GNN and GMP solution also degrade as the velocity of the robots is increased, it is observed that having memory significantly helps the system be more robust to violating the bounding box constraints.

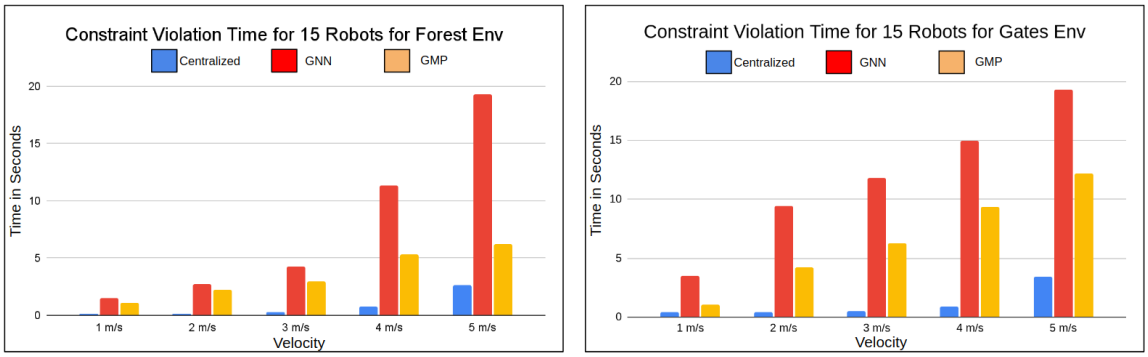


Figure 5.8: Constraint violation in Seconds vs Velocities. Left Constraint Violation in seconds for 15 robots for goals at a distance of 100m. **Right** Constraint violation in seconds for 15 robots for a course consisting of 8 checkpoints for a total distance of 100m.

5.4 DISCUSSION AND GUIDING IDEAS

This chapter introduces a novel methodology called Graph Memory Policies (GMP) that computes decentralized perception-action-communication loops for teams of robots. GMP leverages the use of CNNs to perceive the world and uses a novel graph recurrent network architecture to encode past history while computing decentralized local policies. GMP opens the door for many multi-robot collaborative applications that need to rely only on decentralized information such as network coverage. In the next chapter, we shall leverage GMP to build distributed large scale solutions for multi-robot coverage, where a team of robots is tasked with following another team of robots by relying only on their visual sensors and local communication.

6 LEARNING DECENTRALIZED PERCEPTION ACTION COMMUNICATION LOOPS FOR MULTI ROBOT COVERAGE

6.1 INTRODUCTION

In Chapter 5, we introduced Graph Memory Policies to enable teams of robots to accomplish tasks collaboratively. However, most of the focus in the preceding chapter was on a relatively simple task of navigating a cluttered space and mantaining a formation. To investigate the ability of Graph Memory Policies to handle scale and complexity, we now consider multi-robot coverage, a modified version of the unlabeled motion planning problem that we first discussed in Chapter 2 and investigate if Graph Memory Policies can offer a viable solution for the multi-robot coverage problem. In the coverage problem considered here there exist a team of *targets* $\mathbf{M}$ and a team of robots $\mathbf{N}$ where $\mathbf{N} \leqslant \mathbf{M}$ and there exists no prior matching between robots and targets. The targets evolve according to some unknown dynamics at each time step and the task of the robot team is to cover as many robots as possible. In this work, we define coverage as a function of distance between targets and robots and assume one robot can cover up to five targets at a given time.

As in Chapter 5, the robots are only equipped with a front facing camera and an inertial measurement unit (IMU). We choose this problem as there exist several interesting ideas

to investigate here. In the first version of the coverage problem, we initialize robots close to targets and have the robots follow the targets around a structured environment. We call this version of the multi-robot coverage problem **Follow**. In the Follow problem, robots must adapt to sudden changes in behaviors of the target, for example groups of targets might split up or come together in which case the robots must be able to adapt their behaviors such that maximum coverage is achieved. By proposing a solution that utilizes both Convolutional Neural Networks (CNNs) to detect cars, and Graph Neural Networks with memory (GMP), we show that with the GMP solution, robot teams with good initialization are able to closely follow and cover targets even when they exhibit behaviors not seen during training.

In the second version of the problem, robots and targets are initialized randomly and the robots are tasked with finding and covering as many targets as possible. We call this flavor the coverage problem **Fetch**. This problem is interesting because it leverages the local information flow through the graph of robots and forces robots that do not see any targets to aggregate to robots that do see targets or move away from targets that are already covered in the hope of finding new uncovered targets such that the team wide objective of covering as many targets as possible is accomplished.

In the problems considered in the previous chapter(s), the solutions were often able to transfer easily after training on a small number of robots and testing on a larger number of robots. However, in the problems considered here that is not often the case. For example, consider the **Fetch** problem where robots must explore the space to find the targets. In this setting, the policy trained for say ten robots to find ten targets does not work when attempting to infer on twenty robots attempting to search for twenty targets. The reason being that when the twenty robots find the first ten targets, the task (for which the policy was originally trained) is accomplished. One solution is to simply retrain the team of twenty robots to cover twenty targets. Such an approach can get cumbersome

as training with a larger number of robots needs more training samples. However, we show that when the policies are trained on a sufficiently large number of robots, it is able to transfer to an even larger number of robots without requiring re-training. We draw on key results from graphon signal processing [66] to explain large scale transfer. A graphon is a bounded symmetric measurable function $\mathbf{W} : [0,1]^2 \rightarrow [0,1]$ that can be thought of as an undirected graph with an uncountable number of nodes. This can be seen by relating nodes i and j with points $u_i, u_j \in [0,1]$ and edges (i,j) with weights $\mathbf{W}(u_i, u_j)$. The theorem in [66] shows that there exist bounds on the transferability of graph filters between two deterministic graphs sampled from the same graphon. This result can be leveraged to demonstrate transfer of policies for a complex behavior such as searching for a large number of targets from a large number of robots (for example 50 robots searching for 50 targets) to an even larger number of robots (for example 100 robots searching for 100 targets) with small loss in performance behavior as both the 50 robot graph and the 100 robot graph can be thought of as deterministic graphs belonging to the same "graphon family".

6.2 METHODOLOGY

6.2.1 Preliminaries

We pose the problem of learning multi-robot coverage for a team of robots as a policy learning problem in a collaborative Markov team [24]. The team is composed of $\mathbf{N}$ robots generically indexed by n which at any given point in time t occupy a position or pose $p_{nt} \in \mathbb{R}^3$ in configuration space and must choose an action $\mathbf{a}_{nt} \in \mathcal{A}$ in action space. For the sake of notation, we drop the time index unless otherwise specified. Let the targets

consist of $\mathbf{M}$ targets generically indexed by m which at any given point in time t occupy a position or pose $t_{mt} \in \mathbb{R}^2$. The positions of the targets evolve according to some unknown dynamics f; i.e $t_{mt+1} = f(t_{mt})$. The rest of the setup for the team of robots follows that of Chapter 5.

At each time step robot n has access to an image I_n from a front facing camera. To simulate communication, we also assume that robot n also has a fixed communication radius ϵ and can communicate with at most $k(k << \mathbf{N})$ nearest robots inside ϵ. Let the information from these k nearest robot be represented as $\boldsymbol{\nu}_n = \{\nu_1, \ldots, \nu_k\}$ where ν_k is some information communicated by robot k to robot n. It is important to note that this set of k nearest neighbors is *not fixed* since the robots are evolving in space and time and the set of k robots closest to n can change from one timestep to another. Thus the state of robot n at any given time can be given as $\mathbf{x}_n = [I_n, \boldsymbol{\nu}_n]$.

Given access to this robot state $\mathbf{x}_n$, robot n chooses an e-dimensional action $\mathbf{a}_n \in \mathbb{R}^e$ that controls the pose of the robot according to some unknown underlying dynamical model. To best capture the stochastic nature of the dynamics model, the robot samples *continuous* actions $a_n := \pi_n(\mathbf{x}_n|\mathbf{a}_n)$ from a policy π_n. Collectively, for the robot swarm we denote $\Pi = [\pi_1 \ldots \pi_N]$. Additionally, the team and environment are assumed to be Markovian. This means that if one were to collect the robot configurations in the vector $\mathbf{x}_t := [\mathbf{x}_{1t}, \ldots, \mathbf{x}_{Nt}]^\top$ and actions in the vector $\mathbf{a}_t := [\mathbf{a}_{1t}, \ldots, \mathbf{a}_{Nt}]$ then, the evolution of the system is completely determined by some conditional transition probability $\mathcal{T}(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t)$.

The team of robots is tasked with covering as many targets as possible in minimum time and also avoid collisions with other robots and other entities in the environment. We measure the target coverage as a utility function or reward function:

$$r_t = \sum_{i=1}^{M} \mathbb{I}\{\min_m \|p_{nt} - t_{mt}\| \leqslant \delta\} \tag{6.1}$$

where $\mathbb{I}$ is the indicator function. In the case that all the targets are covered, $r_t = |\mathbf{M}|$. The objective for the whole team is then to compute actions for all $\mathbf{a}_t := \Pi(\mathbf{a}_t|\mathbf{x}_t)$ such that the expected sum of rewards over some time horizon T is maximized:

$$\max_{\theta} \mathbb{E}_{\Pi}\left[\sum_{t}^{T} r_t\right] \tag{6.2}$$

where θ are the parameters of Π. In the next section, we introduce our modular approach we demonstrate how Graph Memory Policies can be used to learn Π that which consists of learning an offline perception system and learning a distributed control policy scheme with limited communication with nearby robots. The only difference between the problem considered here and the problem considered in the Chapter 5 is the design of vision system. A snapshot of the problem considered in this paper can be visualized in fig 6.1.

Figure 6.1: Decentralized Perception Coverage by Robot Teams A team of aerial robots is tasked with covering a team of targets. On the left a subset of targets are detected by a robot (bounding boxes annotated in blue) and on the right a different subset of targets are detected by another robot (annotated in orange). The team of robots must collaboratively cover as many targets as possible. The targets are moving and the aerial robots have no prior about how the targets are going to evolve in space and time.

6.3 PERCEPTION SYSTEM

6.3.1 Dataset Generation

We use the AirSim ([50]) simulator for all of our experiments. To generate the dataset for training the object detection model to be used on each of the quadrotors, we employ a setup with one moving quadrotor following behind a car target as seen in fig 6.1 as the two navigate a number of waypoints in the environment. Additionally, many stationary cars in various orientations lie on the paths traversed by the single moving quadrotor and car. The quadrotor captures 640x480 resolution (configurable) images of the roads with a single front-facing camera tilted 45 degrees downward. We add a number of disturbances to the quadrotor's waypoints in order to give it an erratic path that leads to a variety of different positions of the target car in the quadrotor's images. In this manner, we capture 3748 images. Out of these, 2001 were annotated for use in training. We augment the dataset by using standard practices in computer vision such as mirroring the images and adding small amounts of skew to result in a dataset that has about roughly 6000 training images. In order to be able to adapt to a variety of scenarios, we employ domain randomization where parameters such as the background textures and colors are randomized. A snapshot of some of our training images can be seen in Fig 6.2

6.3.2 Training

We use an off-the-shelf object detection model on each of the n robots. Specifically, we utilize a tiny Yolo-v3 model [67]. The network takes as input 640x480 images and outputs feature vectors of shape 4500x6. Each row of this feature vector corresponds to the output

Figure 6.2: Dataset Generation for Perception System.

of a particular anchor box (i.e. there are 4500 anchor boxes in total). Of these anchors, 900 correspond to the 15x20 grid of larger anchor cells, and 3600 correspond to the 30x40 grid of smaller anchor cells. Each anchor cell corresponds to 3 anchor boxes with different shape priors. The predictions g_i of a given anchor box i are made relative to that anchor's prior information and then transformed into global image coordinates. The components of a row g_i of the output feature vector are outlined below:

$$g_i = \begin{pmatrix} x_i & y_i & w_i & h_i & o_i & c_i \end{pmatrix} \tag{6.3}$$

o_i- Objectness score for a given anchor.

c_i- Class probabilities for a given anchor.

x_i- x-coordinate of anchor's predicted box center.

y_i- y-coordinate of anchor's predicted box center.

w_i- Width of the predicted box.

h_i- Height of the predicted box.

Following [68], each anchor initially predicts bounding box location, width, and height relative to its own position in the anchor grid as well as its own prior shape. However, these intermediate outputs are transformed to give the final x_i, y_i,, w_i, and h_i. While training we look to optimize the following multi-part loss function:

$$\begin{aligned} &\sum_{i=1}^{4500} 1_i^{obj}(o_i - \hat{o}_i)^2 + \\ &\lambda_{noobj} \sum_{i=1}^{4500} (1 - 1_i^{obj})(o_i - \hat{o}_i)^2 + \\ &\lambda_{coord} \sum_{i=1}^{4500} 1_i^{obj}(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + \\ &\lambda_{coord} \sum_{i=1}^{4500} 1_i^{obj}(w_i - \hat{w}_i)^2 + (h_i - \hat{h}_i)^2 \\ &\lambda_{noobj} = 0.5,\ \lambda_{coord} = 5 \end{aligned} \tag{6.4}$$

We begin by pre-training our tiny YOLO-v3 architecture on the COCO dataset. From there, we fine-tune on our constructed dataset as seen in Fig 6.2. At inference time, we pick the top k anchor boxes as relevant objects detected in the scene and pass the information to the control subsystem. In this work, we set k to be 5.

In this work, while we use all components of g_i to train the perception system, we only pass the dimensions of the bounding box to the control system, i.e for a given output feature vector g_i, we only pass w_i and h_i to the control system. The reason for this being that these are the only two dimensions in the output feature that are permutation invariant

and can be used to exploit permutation invariance in the GNN. Thus, the final output of the perception system for robot n at any given time t is given as

$$\hat{g}_{nt} = \Big[[w_1, h_1], \ldots, [w_k, h_k]\Big] \tag{6.5}$$

6.3.3 Control System

We use the same graph memory network from Chapter 5 to build our decentralized control system. We recommend skipping this section if the reader is already familiar with it from Chapter 5.

Consider a graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ described by a set of $\mathbf{N}$ nodes denoted $\mathbf{V}$, and a set of edges denoted $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$. This graph is considered as the support for a data signal $\mathbf{x}_t = [x_{1t}, \ldots, x_{Nt}]^\top$ where the value x_{nt} is assigned to node n at time t. The relation between $\mathbf{x}_t$ and $\mathcal{G}$ is given by a a matrix $\mathbf{S}$ called the graph shift operator. The elements of $\mathbf{S}$ given as s_{ij} respect the sparsity of the graph, i.e $s_{ij} = 0$, for all $i \neq j$ and $(i, j) \notin \mathbf{E}$.

Valid examples for $\mathbf{S}$ are the adjacency matrix, the graph laplacian, and the random walk matrix. $\mathbf{S}$ defines a map $\mathbf{y}_t = \mathbf{S}\mathbf{x}_t$ between graph signals that represents local exchange of information between a node and its one-hop neighbors. More concretely, if the set of neighbors of node n is given by $\mathfrak{B}_n$ then $y_{nt} = [\mathbf{S}\mathbf{x}_t]_n = \sum_{j=n, j\in\mathfrak{B}_n} s_{nj} x_{nt}$. This operation performs an aggregation of data at node n from its neighbors that are one-hop away at time t. The aggregation of data at all nodes in the graph is denoted $\mathbf{y}_t = [y_{1t}, \ldots, y_{Nt}]$. By repeating this operation, one can access information from nodes located further away. For

example, $\mathbf{y}_t^k = \mathbf{S}^k\mathbf{x}_t = \mathbf{S}(\mathbf{S}^{k-1}\mathbf{x}_t)$ aggregates information from its k-hop neighbors. Now one can define the spectral K-localized graph convolution at time t as :

$$\mathbf{z}_t = \sigma \sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{x}_t = \sigma \mathbf{H}(\mathbf{S})\mathbf{x_t} \tag{6.6}$$

where $\mathbf{H}(\mathbf{S}) = \sum_{k=0}^{K} h_k \mathbf{S}^k$ is a linear shift invariant graph filter [45] with coefficients h_k, K is a user set parameter that defines how many hops we would like to aggregate information over and similar to CNNs the output of the GCN is fed into a pointwise non-linear function σ. To introduce memory at each node, we leverage the fact that the output of a graph neural network is also a graph in itself. Thus, we assume that the state $\mathbf{z}_t$ is in itself a hidden nodal state. By defining another linear shift operator, $\mathbf{B}(\mathbf{S}) = \sum_{k=0}^{K} b_k \mathbf{S}^k$ with coefficients b_k, we can write the graph memory neural network (GMN) output as

$$\mathbf{z}_t = \sigma(\mathbf{H}(\mathbf{S})\mathbf{x_t} + \mathbf{B}(\mathbf{s})\mathbf{z_{t-1}}) \tag{6.7}$$

This representation is similar to a recurrent neural network where a hidden state is updated over a sequence while being provided with an input sequence and the output at any given time t is computed by a linear combination of the hidden state at time t and the sequence input at time t. A visual representation of a graph memory neural network can be seen in Fig 6.3, bottom. In order to learn a Π that maximizes the reward in Eqn. 6.2, we propose parameterizing Π with a Graph Memory Neural Network (GMN). More formally, let the state of robot n at time be given as $\mathbf{x}_n t = \mathcal{F}(I_{nt})$, i.e the output from the perception system. The robots can be represented as a graph $\mathcal{G}$ with $\mathbf{N}$ nodes where each robot represents a node in the graph and the edges are based on proximity relationships. This graph acts as the support for the data vector $\mathbf{x}_t = [\mathbf{x}_{1t}, \ldots, \mathbf{x}_{Nt}]$. To compute the policies, a GMN

architecture with L layers is initialized. At each layer according to Eq.6.7, the output is given as:

$$\mathbf{z}_t^{l+1} = \sigma\big(\mathbf{H}(\mathbf{S})\mathbf{z}_t^{l-1} + \mathbf{B}(S)\mathbf{z}_{t-1}^{l}\big) \tag{6.8}$$

where σ is a pointwise non-linear function, $\mathbf{z}_t^0 = \mathbf{x}_t$ and $\mathbf{z}^L = \Pi = [\pi_1, \ldots, \pi_N]$. In practice, the final layer outputs are parameters of Gaussian distributions from which actions are sampled. Intuitively at every node, the GMN architecture aggregates information and uses this information to compute policies. While rolling out a trajectory at each timestep t each robot receives the same centralized reward r_t (defined in Sec 5.2 and Sec 5.3) and attempts to learn a policy that best optimizes this reward. It is assumed that the policies for the robots are independent. The overall objective function function for all the robots as given in Eq.6.2

$$J = \sum_{n=1}^{N} \max_{\theta} \mathbb{E}_{\Pi}\Big[\sum_{t}^{T} r_t\Big] \tag{6.9}$$

where θ now represents the filter weights of the GMN for Π. Consider a trajectory $\tau = (\mathbf{x_0}, \mathbf{a_0}, \ldots, \mathbf{x_T}, \mathbf{a_T})$. Since the reward along a trajectory is the same for all robots and all robot policies are assumed independent, using direct differentiation the policy gradient is given as :

$$\begin{aligned}\nabla_\theta J = \mathbb{E}_{\tau\sim(\pi_1,\ldots,\pi_N)}\Bigg[\Big(\sum_{t=1}^{T} \nabla_\theta \log[\pi_1(\mathbf{a}_{1t}|\mathbf{x}_{1t}) \\ \ldots \pi_N(\mathbf{a}_{Nt}|\mathbf{x}_{Nt})]\Big)\Big(\sum_{t=1}^{T} r_t\Big)\Bigg]\end{aligned} \tag{6.10}$$

The weights θ of the GMN, are then updated using any variant of stochastic gradient descent. We call this algorithm Graph Memory Policies (GMP). An overview of the modified GMP system for the multi-robot coverage task can be seen in Fig

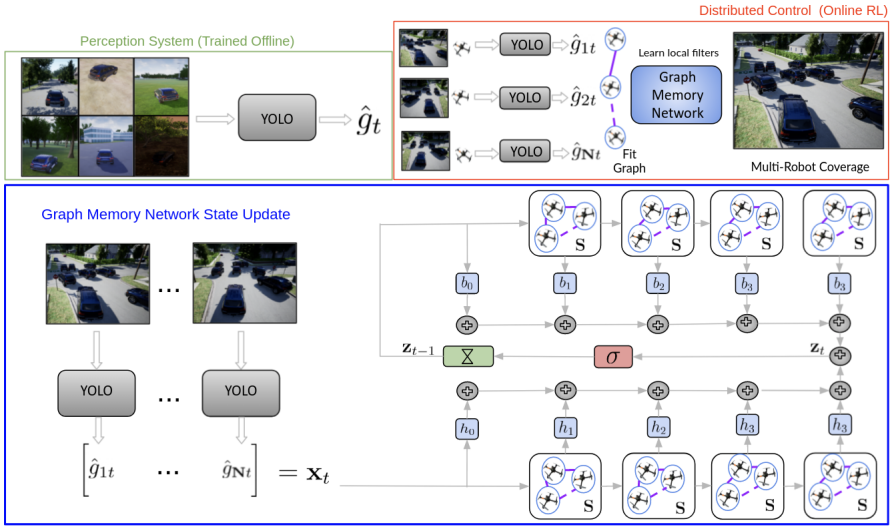


Figure 6.3: Modular Approach for Multi-Robot Coverage The proposed decentralized solution to co-ordinate a team of robots that follows a set of targets consists of two subsystems. **Top L** The perception subsystem is trained offline for a single robot to predict bounding boxes for targets in the environment that the robot must cover. **Top R** The control subsystem uses a graph memory neural network that takes in the prediction from the vision subsystem and communicates with nearby robots to collaboratively cover as many targets as possible. **Bottom** State computation in a single layer of a graph memory network with $K = 4$. The blue blocks represent linear weights, red blocks represent non-linearity and the green block represents a time shift. We stack multiple such layers and the output of the final layer is Π.

6.4 FOLLOW

In this task, the team of robots must follow a team of targets around a simulated city block environment. The key assumption in this task is that the team of robots is initialized close to the targets such that at least one of the targets is within the camera range of one of the robots. In the next section we shall relax this assumption. The reason being that the follow behavior requires a significantly different cost function from that described in Eqn. 6.2. In practice we also add additional terms to the cost function to penalize robots for colliding into each other. Further, to make the problem a little more tractable, the robots are constrained to $\mathbb{R}^2$ instead of $\mathbb{R}^3$.

For this task during training we train on a small number of robots and transfer to a larger number of robots during inference. Training on a larger swarm in this scenario offers very little benefits during inference time. Additionally, we also model target behaviors such as splitting up and regrouping but do not necessarily train on these models, instead choosing to directly test on them during inference. In this work, the velocities of the targets and the robots are kept the same. In the case that the targets are significantly faster than the robots following them, the problem produces intractable results. A qualitative snapshot of our results can be seen in Fig 6.4

Figure 6.4: Multi-Robot Coverage On the top, the figure on the left represents a top down view of our environment in which we test our robots. Targets are constrained to move on the road while robots are free to move anywhere 5 meters above the ground plane. On the bottom, we visualize the behavior of the robots over time (left to right) and observe the robots are able to split up and cover the targets even when the robots have not been trained to cover splitting behaviors in the targets.

In addition to the qualitative results, we also demonstrate a quantitative plot for differing number of targets (cars) vs. targets which can be seen in Fig 6.5 We observe that even as

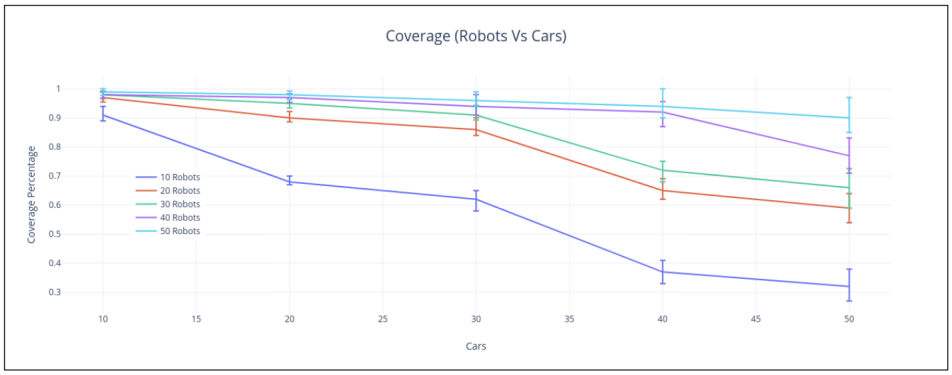


Figure 6.5: Coverage Percentage for Robots Vs. Cars

the number of target cars are increased the robots are able to co-ordinate behaviors by splitting into groups to best cover as many target cars as possible and as such can conclude that GMP offers a viable solution for multi-robot coverage where robots are equipped only with a front facing camera and an IMU while making no assumptions about the dynamics of robots or cars or structure of the environment.

6.5 FETCH AND LARGE SCALE TRANSFERANCE WITH GRAPHONS

In the last section, we demonstrated a *follow* behavior where robots collaboratively cover a team of targets around a simulated environment. One of the assumptions in the follow behavior was that the robots have a good initialization, i.e the robots are initialized close to the targets such that atleast one of the target cars are within the visual frame of one of the robots. The reason being that the loss function used for the follow behavior trasnfers very well from a small number of robots to a larger number of robots since the cost function is invariant to the number of targets operating in the environment. In this section, we consider a slightly different problem, the robots are randomly initialized in the space as are

the targets without any assumptions about the initialization. This problem is significantly different from the unlabeled motion planning problem where the robots were given a representation of nearest goals. Instead in this problem the robots are only presented with an image representation of their environment which may or may not have a target car detected.

Now in this setting, the robots must explore the environment to find targets and either do one of two things; co-ordinate neighboring robots to navigate to their position if the number of targets and/or communicate with other robots to continue searching the environment for other targets. However, this cost function is no longer invariant to the number of targets operating in the environment; i.e a cost function that optimizes search for 10 robots is different from a cost function that optimizes search for say 50 robots. In such a setting, our policies no longer transfer after training from a small number of robots to a larger number of robots.

However, to overcome this difficulty in transferring policies, we instead look to draw recent results from the field of graphon signal processing [66]. A graphon is a bounded symmetric measurable function $\mathbf{W} : [0,1]^2 \to [0,1]$ that can be thought of as an undirected graph with an uncountable number of nodes. This can be seen by relating nodes i and j with points $u_i, u_j \in [0,1]$ and edges (i,j) with weights $\mathbf{W}(u_i, u_j)$. This construction allows us to have a limit object interpretation for $\mathbf{W}$, letting us define a sequence of graphs $\{\mathcal{G}_n\}_{n=1}^{\infty}$ that converge to $\mathbf{W}$ [69]. We briefly explain how one can relate multiple graphs through a graphon in the next two sections.

Graph Homomorphisms

Consider a graph $\mathcal{G}$ as a set of vertices $\mathbf{V}(\mathcal{G})$ and a set of edges $\mathbf{E}(\mathcal{G})$ between the vertices (excluding loops and multiple edges). Similarly define another graph $\mathcal{H}$ with vertices $\mathbf{V}(\mathcal{H})$ and edges $\mathbf{E}(\mathcal{H})$. A graph homomorphism from a different graph $\mathcal{H}$ to graph $\mathcal{G}$ is

defined as a map from $\mathbf{V}(\mathcal{H})$ to $\mathbf{V}(\mathcal{G})$ that preserves edge adjacency; i.e for every edge $\{v, w\}$ in $\mathbf{E}(\mathcal{H})$ the edge $\{\psi(v), \psi(w)\}$ exists in $\mathbf{E}(\mathcal{G})$. The total number of possible maps from $\mathcal{H}$ to $\mathcal{G}$ can be given as $|\mathbf{V}(\mathcal{G})|^{|\mathbf{V}(\mathcal{H})|}$. However, it is possible that not all of these maps are homomorphisms/only some of them are. Let the number of homomorphisms between $\mathcal{H}$ and $\mathcal{G}$ be denoted by $\hom(\mathcal{H}, \mathcal{G})$

We define homomorphism *density* of $\mathcal{H}$ into $\mathcal{G}$ as

$$t(\mathcal{H}, \mathcal{G}) = \frac{\hom(\mathcal{H}, \mathcal{G})}{|\mathbf{V}(\mathcal{G})|^{|\mathbf{V}(\mathcal{H})|}} \tag{6.11}$$

Intuitively, $t(\mathcal{H}, \mathcal{G})$ gives us the probability that a randomly chosen map from $\mathbf{V}(\mathcal{H})$ to $\mathbf{V}(\mathcal{G})$ preserves edge adjacency.

Graphons as graph limit objects

It is possible to define sequences of graphs $\{\mathcal{G}_n\}_{n=1}^{\infty}$ where $n = |\mathbf{V}(\mathcal{G})|$ is the number of nodes in $\mathcal{G}$. The graphon homomorphism can be given similar to those for graphs. Let $t(\mathcal{F}, \mathbf{W})$ be the density of homomophisms of graph $\mathcal{F}$ into graphon $\mathbf{W}$, then a sequence of $\{\mathcal{G}_n\}$ converges to the graphon $\mathbf{W}$ if, for all finite unweighted and undirected graphs $\mathcal{F}$;

$$\lim_{n\to\infty} t(\mathcal{F}, \mathcal{G}_n) = t(\mathcal{F}, \mathbf{W}) \tag{6.12}$$

Thus, a graphon identifies a collection of graphs and regardless of the size of the graphs, these graphs can be considered *similar* in the sense that they belong to the same graphon family. The work of [66] builds on this result and shows that if two graphs are similar to each other in the sense that they belong to the same graphon family, then the GNNs trained on each of the graphs also exhibit some degree of transferability (or similarity). More formally, the theorem in [66] states that:

Theorem 3. ***Ruiz. et al*** *Let $\phi(\mathcal{G})$ be a GNN with fixed parameters. Let $\mathcal{G}_{n1}$ and $\mathcal{G}_{n2}$ be deterministic graphs with n_1 and n_2 nodes obtained from a graphon* **W**. *Then, under mild conditions,*

$$\|\phi(\mathcal{G}_{n_1}) - \phi(\mathcal{G}_{n_2}\| = \mathcal{O}(n_1^{-0.5} + n_2^{-0.5})$$

We point the reader to [66] for a full proof of the theorem. We utilize this theorem to construct results for the *fetch* behavior for large teams of robots. Consider a graph $\mathcal{G}_{n_1}$ of robots constructed by considering the n_1 robots as nodes of the graph and edges populated on distance based rules. Similarly consider another graph $\mathcal{G}_{n_2}$ consisting of n_2 robots. Theorem 3 tells us that the GNN trained for n_1 robots can transfer (mildly) to the GNN trained on n_2 robots if both graphs $\mathcal{G}_{n_1}$ and $\mathcal{G}_{n_2}$ belong to the same family of graphs generated by an underlying graphon. Since the robots themselves are all homogeneous and unchanged (i.e the nodes of the graphs are unchanged) and the local topology of the graph remains unchanged (i.e edge connectivity rules are same for both graphs), we can informally hypothesize that both robot graphs $\mathcal{G}_{n_1}$ and $\mathcal{G}_{n_2}$ belong to the same graphon family and thus, the GNN policies trained on one of them must work well within some bounds on the other. A rigorous mathematical proof would require proving the graphon signal formed by an infinite number of robots is continuous and would also require extending the result of [66] to a graph memory network and is beyond the scope of this thesis and we leave this for future work. We empirically verify our hypothesis by looking at the qualitative and quantitative results for the *fetch* behavior below in Fig **??** and Fig 6.7 respectively.

We observe from Fig 6.7 that at inference time the policy trained on the larger number of robots when transferred over to an even larger number at inference time, the dropoff in performance starts reducing as expected from Theorem 3. It is important to note here that

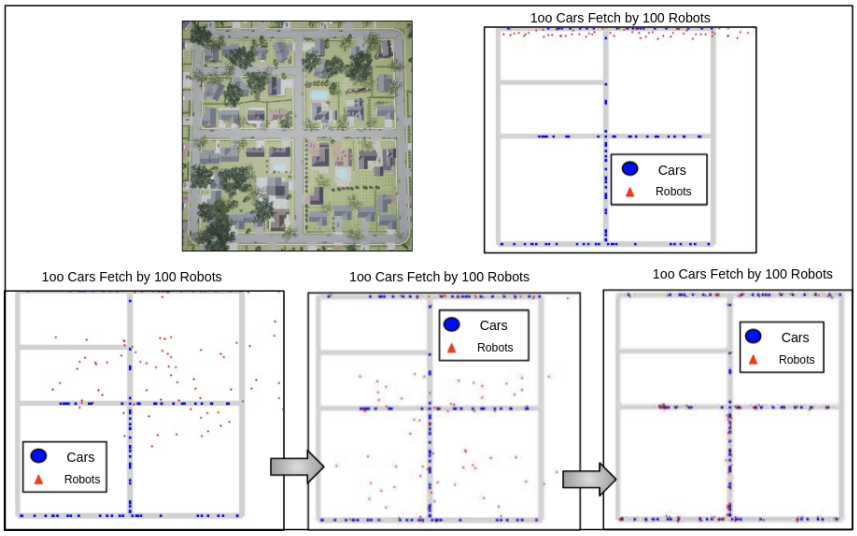


Figure 6.6: Fetch Behavior for Multi-Robot Coverage (Qualitative) On the top, the figure on the left represents a top down view of our environment in which we test our robots. Targets are constrained to move on the road while robots are free to move anywhere 5 meters above the ground plane. On the bottom, we visualize the behavior of the robots over time (left to right) and observe the robots are able to cover most of the targets during inference.

this result is not just valid to the *fetch* problem considered here but in fact applies almost all problems considered in this body of work. Other problems considered in this work such as collaborative flying or co-operative unlabeled motion planning were designed to be invariant to the number of robots by careful design of each robots state space and the cost function and as such the policy trained on a smaller number of robots would easily transfer over to a larger number of robots at inference time. In the *fetch* problem, while the state space of each robot is still permutation invariant, the cost is no longer permutation invariant. As such, the results here can be applied to other problems considered in this thesis without having to carefully design a cost function.

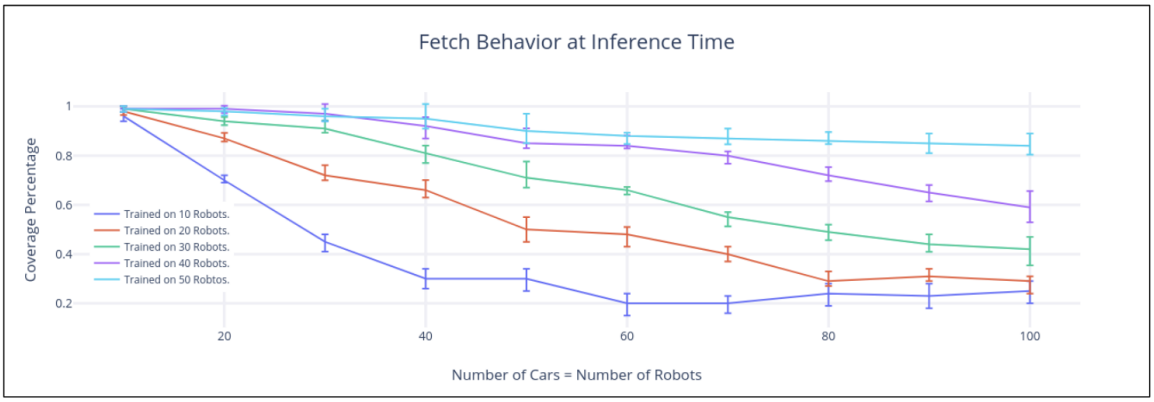


Figure 6.7: Fetch Behavior for Multi-Robot Coverage (Quantitative Results)

6.6 DISCUSSION AND GUIDING IDEAS

In this chapter, we investigate another body of problems namely the multi-robot coverage problem and a modified version of the problem that we call *fetch*. In both versions of the problem the robots have access only to their camera image and can communicate with nearby neighbors. For the coverage problem, we are able to demonstrate good performance with the graph memory network and can demonstrate the behaviors transfer to a larger number of robots even when the number of targets at inference time are increased. Further, the graph memory network is able to learn policies that can adapt to behaviors in the target cars at inference time such as splitting and regrouping which are not seen during training. The *fetch* problem increases the complexity of the problem posed to the graph memory network due to the non-ease in designing a permutation invariant cost function. Due to this limitation, when looking to scale to a large number of robots, we would have to retrain our policy thus bringing us back to some of the issues with training a large number of robots at the same time as discussed in Chapter 2. Instead, we show that the GNN results can still be adapted to a larger number of robots without having to train on it due to the existence of an underlying graphon that relates the smaller sized robot graph at

train time to the much larger robot graph at test time. While this represents another step forward in designing large scale model free solutions for robot teams, a mathematically rigorous proof for the existence of the underlying graphon tying the two graphs is still required.

7 CONCLUSION

In this work, we introduce the use of model free decentralized solutions for multi-robot problems as a way of generating approximate solutions for otherwise intractable problems. A caveat of model-free solutions is the need for a large number of samples to learn a solution. This problem is compounded when we wish to look at model-free learning solutions for large robot teams. In machine learning, this is the well known *curse of dimensionality*. However, in this work we demonstrate that by leveraging repeating local structures in large teams of robots and by thoughtful design of policy parametrizations and cost functions we can learn decentralized solutions for teams of robots to achieve a wide variety of tasks.

In Chapter 2, we investigate the use of vanilla off the shelf model free algorithms to demonstrate feasibility of learning a decentralized solution by using a global cost function. This approach while yields good results, is extremely limited in its scope and cannot be extended to a larger number of robots. One of the significant ideas or innovation(s) in this work, is to instead use a graph neural network to parameterize the policy representation for the robots to leverage repeating local structure among the robots. The use of a graph neural network requires a careful design of the state space for each robot and the cost function to ensure that both are permutation invariant. Over Chapter 3 and 4 we analyze the efficacy of using a graph neural network without for collaborative control and unlabeled motion planning albeit assuming perfect state information and instantaneous lossless communication. In Chapter5 we further increase the complexity of the problem by

introducing a noisy perception sensor as the input to the policy network. Additionally, we demonstrate the need for memory in real world situations when coordinating robots faced with noisy perception and fast motion through environments that require non-reactive behaviors. Lastly, in chapter 6, we drop the need for a permutation invariant cost function and demonstrate that our key ideas of transferring results between a small number of robots at training time and a larger number of robots during inference time still hold due to the existence of an underlying relationship between the two robot graphs.

While the science of using graph neural networks and machine learning to design solutions for multi-robot teams is still in its infancy, we hope that future research can build on ideas from graph neural networks to design complex policies that can be executed by large swarms of low-cost robots to automate hazardous tasks such as construction (unlabeled motion-planning), forest fire-fighting (navigating a dense cluttered outdoor environment with limited perception and communication), window washing (coverage), at an economical cost.

BIBLIOGRAPHY

[1] Javier Alonso-Mora, Stuart Baker, and Daniela Rus. "Multi-robot navigation in formation via sequential convex programming." In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE. 2015, pp. 4634–4641.

[2] Jaydev P Desai, James P Ostrowski, and Vijay Kumar. "Modeling and control of formations of nonholonomic mobile robots." In: *IEEE transactions on Robotics and Automation* 17.6 (2001), pp. 905–908.

[3] John Enright and Peter R Wurman. "Optimization and Coordinated Autonomy in Mobile Fulfillment Systems." In: 2011.

[4] Ross A Knepper, Todd Layton, John Romanishin, and Daniela Rus. "Ikeabot: An autonomous multi-robot coordinated furniture assembly system." In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 855–862.

[5] J. Stephan, J. Fink, V. Kumar, and A. Ribeiro. "Concurrent Control of Mobility and Communication in Multirobot Systems." In: *IEEE Transactions on Robotics* 33.5 (2017), pp. 1248–1254.

[6] David Saldana, Reza Javanmard Alitappeh, Luciano CA Pimenta, Renato Assunçao, and Mario FM Campos. "Dynamic perimeter surveillance with a team of robots." In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 5289–5294.

[7] Kiril Solovey and Dan Halperin. "On the hardness of unlabeled multi-robot motion planning." In: *The International Journal of Robotics Research* 35.14 (2016), pp. 1750–1759.

[8] Shayegan Omidshafiei, Ali-Akbar Agha-Mohammadi, Christopher Amato, and Jonathan P How. "Decentralized control of partially observable markov decision processes using belief space macro-actions." In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 5962–5969.

[9] Paul Spirakis and Chee K Yap. "Strong NP-hardness of moving many discs." In: *Information Processing Letters* 19.1 (1984), pp. 55–59.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing atari with deep reinforcement learning." In: *arXiv preprint arXiv:1312.5602* (2013).

[11] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning." In: *International Conference on Machine Learning*. 2016, pp. 1928–1937.

[12] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. "End-to-end training of deep visuomotor policies." In: *arXiv preprint arXiv:1504.00702* (2015).

[13] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. "Curiosity-driven Exploration by Self-supervised Prediction." In: *arXiv preprint arXiv:1705.05363* (2017).

[14] Arbaaz Khan, Clark Zhang, Nikolay Atanasov, Konstantinos Karydis, Vijay Kumar, and Daniel D Lee. "Memory augmented control networks." In: *International Conference on Learning Representations* (2018).

[15] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. "Cognitive mapping and planning for visual navigation." In: *arXiv preprint arXiv:1702.03920* (2017).

[16] Aviv Adler, Mark De Berg, Dan Halperin, and Kiril Solovey. "Efficient multi-robot motion planning for unlabeled discs in simple polygons." In: *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 1–17.

[17] Patrick MacAlpine, Eric Price, and Peter Stone. "SCRAM: Scalable collision-avoiding role assignment with minimal-makespan for formational positioning." In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.

[18] Jingjin Yu and M LaValle. "Distance optimal formation control on graphs with a tight convergence time guarantee." In: *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*. IEEE. 2012, pp. 4023–4028.

[19] Matthew Turpin, Nathan Michael, and Vijay Kumar. "Capt: Concurrent assignment and planning of trajectories for multiple robots." In: *The International Journal of Robotics Research* 33.1 (2014), pp. 98–112.

[20] Jingjin Yu and Steven M LaValle. "Multi-agent path planning and network flow." In: *Algorithmic foundations of robotics X*. Springer, 2013, pp. 157–173.

[21] Matthew Turpin, Kartik Mohta, Nathan Michael, and Vijay Kumar. "Goal assignment and trajectory planning for large teams of interchangeable robots." In: *Autonomous Robots* 37.4 (2014), pp. 401–415.

[22] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. "Multi-agent actor-critic for mixed cooperative-competitive environments." In: *Advances in Neural Information Processing Systems*. 2017, pp. 6379–6390.

[23] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. "Counterfactual multi-agent policy gradients." In: *arXiv preprint arXiv:1705.08926* (2017).

[24] Michael L Littman. "Markov games as a framework for multi-agent reinforcement learning." In: *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 157–163.

[25] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." In: *arXiv preprint arXiv:1509.02971* (2015).

[26] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[27] Ming Tan. "Multi-agent reinforcement learning: Independent vs. cooperative agents." In.

[28] Paolo Fiorini and Zvi Shiller. "Motion planning in dynamic environments using velocity obstacles." In: *The International Journal of Robotics Research* 17.7 (1998), pp. 760–772.

[29] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. "Reciprocal n-body collision avoidance." In: *Robotics research*. Springer, 2011, pp. 3–19.

[30] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul Beardsley, and Roland Siegwart. "Optimal reciprocal collision avoidance for multiple non-holonomic robots." In: *Distributed Autonomous Robotic Systems*. Springer, 2013, pp. 203–216.

[31] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms." In: *arXiv preprint arXiv:1707.06347* (2017).

[32] Jur Van den Berg, Ming Lin, and Dinesh Manocha. "Reciprocal velocity obstacles for real-time multi-agent navigation." In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 1928–1935.

[33] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks." In: *arXiv preprint arXiv:1609.02907* (2016).

[34] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. "Distributed prioritized experience replay." In: *arXiv preprint arXiv:1803.00933* (2018).

[35] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures." In: *arXiv preprint arXiv:1802.01561* (2018).

[36] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. "RLlib: Abstractions for Distributed Reinforcement Learning." In: *International Conference on Machine Learning (ICML)*. 2018.

[37] Arbaaz Khan, Clark Zhang, Daniel D Lee, Vijay Kumar, and Alejandro Ribeiro. "Scalable Centralized Deep Multi-Agent Reinforcement Learning via Policy Gradients." In: *arXiv preprint arXiv:1805.08776* (2018).

[38] Javier Alonso-Mora, Paul Beardsley, and Roland Siegwart. "Cooperative collision avoidance for nonholonomic robots." In: *IEEE Transactions on Robotics* 34.2 (2018), pp. 404–420.

[39] Fernando Gama, Antonio G Marques, Geert Leus, and Alejandro Ribeiro. "Convolutional neural network architectures for signals supported on graphs." In: *IEEE Transactions on Signal Processing* 67.4 (2018), pp. 1034–1049.

[40] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. "A comprehensive survey on graph neural networks." In: *arXiv preprint arXiv:1901.00596* (2019).

[41] Fernando Gama, Joan Bruna, and Alejandro Ribeiro. "Stability of Graph Scattering Transforms." In: *arXiv preprint arXiv:1906.04784* (2019).

[42] Matthew Turpin, Nathan Michael, and Vijay Kumar. "Trajectory design and control for aggressive formation flight with quadrotors." In: *Autonomous Robots* 33.1-2 (2012), pp. 143–156.

[43] Matthew Turpin, Nathan Michael, and Vijay Kumar. "Decentralized formation control with variable shapes for aerial robots." In: *2012 IEEE international conference on robotics and automation*. IEEE. 2012, pp. 23–30.

[44] Vijay Kumar and Nathan Michael. "Opportunities and challenges with autonomous micro aerial vehicles." In: *The International Journal of Robotics Research* 31.11 (2012), pp. 1279–1291.

[45] Santiago Segarra, Antonio G Marques, and Alejandro Ribeiro. "Optimal graph-filter design and applications to distributed linear network operators." In: *IEEE Transactions on Signal Processing* 65.15 (), pp. 4117–4131.

[46] Michael M Zavlanos, Ali Jadbabaie, and George J Pappas. "Flocking while preserving network connectivity." In: *2007 46th IEEE Conference on Decision and Control*. IEEE. 2007, pp. 2919–2924.

[47] Brent Schlotfeldt, Dinesh Thakur, Nikolay Atanasov, Vijay Kumar, and George J Pappas. "Anytime planning for decentralized multirobot active information gathering." In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 1025–1032.

[48] Jonathan Ko, Benjamin Stewart, Dieter Fox, Kurt Konolige, and Benson Limketkai. "A practical, decision-theoretic approach to multi-robot mapping and exploration." In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*. Vol. 4. IEEE. 2003, pp. 3232–3238.

[49] Howie Choset. "Coverage for robotics–A survey of recent results." In: *Annals of mathematics and artificial intelligence* 31.1-4 (2001), pp. 113–126.

[50] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. "Airsim: High-fidelity visual and physical simulation for autonomous vehicles." In: *Field and service robotics*. Springer. 2018, pp. 621–635.

[51] Danijar Hafner, James Davidson, and Vincent Vanhoucke. "TensorFlow Agents: Efficient Batched Reinforcement Learning in TensorFlow." In: *arXiv preprint arXiv:1709.02878* (2017).

[52] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. "Multi-agent actor-critic for mixed cooperative-competitive environments." In: *Advances in Neural Information Processing Systems*. 2017, pp. 6382–6393.

[53] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. "Counterfactual multi-agent policy gradients." In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[54] Emilio Parisotto, Soham Ghosh, Sai Bhargav Yalamanchi, Varsha Chinnaobireddy, Yuhuai Wu, and Ruslan Salakhutdinov. "Concurrent Meta Reinforcement Learning." In: *arXiv preprint arXiv:1903.02710* (2019).

[55] Jiechuan Jiang, Chen Dun, and Zongqing Lu. "Graph convolutional reinforcement learning for multi-agent cooperation." In: *arXiv preprint arXiv:1810.09202* (2018).

[56] Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. "Deep Drone Racing: From Simulation to Reality with Domain Randomization." In: *IEEE Transactions on Robotics* (2019). DOI: 10.1109/TRO.2019.2942989.

[57] Lingyan Ran, Yanning Zhang, Qilin Zhang, and Tao Yang. "Convolutional neural network-based robot navigation using uncalibrated spherical images." In: *Sensors* 17.6 (2017), p. 1341.

[58] Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. "Deep Drone Racing: From Simulation to Reality with Domain Randomization." In: *IEEE Transactions on Robotics* (2019). DOI: 10.1109/TRO.2019.2942989.

[59] Antonio Loquercio, Ana I. Maqueda, Carlos R. del Blanco, and Davide Scaramuzza. "DroNet: Learning to Fly by Driving." In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 1088–1095. DOI: 10.1109/LRA.2018.2795643.

[60] D. Mellinger and V. Kumar. "Minimum snap trajectory generation and control for quadrotors." In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 2520–2525. DOI: 10.1109/ICRA.2011.5980409.

[61] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world." In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 23–30.

[62] Arbaaz Khan, Ekaterina Tolstaya, Alejandro Ribeiro, and Vijay Kumar. "Graph Policy Gradients for Large Scale Robot Control." In: *arXiv preprint arXiv:1907.03822* (2019).

[63] Arbaaz Khan, Vijay Kumar, and Alejandro Ribeiro. "Large Scale Distributed Collaborative Unlabeled Motion Planning With Graph Policy Gradients." In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 5340–5347.

[64] Arbaaz Khan, Chi Zhang, Shuo Li, Jiayue Wu, Brent Schlotfeldt, Sarah Y Tang, Alejandro Ribeiro, Osbert Bastani, and Vijay Kumar. "Learning Safe Unlabeled Multi-Robot Planning with Motion Constraints." In: *arXiv preprint arXiv:1907.05300* (2019).

[65] Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. "Search-based motion planning for aggressive flight in se (3)." In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2439–2446.

[66] Luana Ruiz, Luiz Chamon, and Alejandro Ribeiro. "Graphon neural networks and the transferability of graph neural networks." In: *Advances in Neural Information Processing Systems* 33 (2020).

[67] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement." In: *arXiv* (2018).

[68] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[69] Christian Borgs, Jennifer T Chayes, László Lovász, Vera T Sós, and Katalin Vesztergombi. "Convergent sequences of dense graphs I: Subgraph frequencies, metric properties and testing." In: *Advances in Mathematics* 219.6 (2008), pp. 1801–1851.