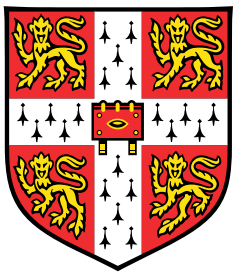
Graph Neural Networks for Multi-Robot Coordination



Qingbiao Li

Supervisor: Dr Amanda Prorok

Department of Computer Science and Technology
University of Cambridge

This dissertation was submitted on Jan, 2023 for the degree of Doctor of Philosophy
Doctor of Philosophy

Hughes Hall July 2023

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Qingbiao Li
July 2023

Dedicate this thesis to my loving parents (Guojun Li and Cuizhao Ye) and sister (Limin Li), and grandma (Julan Liang).

Abstract

Graph Neural Networks for Multi-Robot Coordination

Qingbiao Li

The rapid growth in population and ongoing urbanization yield the need for automatic systems with high productivity and efficiency. Multi-robot systems are developed to respond to this by controlling a team of robots to handle large-scale and complex tasks (i.e. conducting search and rescue operations after earthquakes). The key component of controlling such systems is guiding each robot from its starting place to its goal using a collision-free path in a given environment, called **multi-robot motion planning**. As the size of the team increases, it is getting computationally expensive and intractable to compute the solution based on the centralized approach. Therefore, researchers have been investigating decentralized approach to compute trajectories for each robot separately, and re-planning only in case of conflicts. This method can minimise the task's computing complexity, but it is prone to producing suboptimal and partial solutions. Balancing **optimality** and **completeness** guarantees of computing a solution is still an open problem.

In this thesis, we are particularly interested in investigating machine learning (especially graph neural network) based approaches to find the **trade-off between optimality and complexity** by **offloading online computation into an offline training process**. Yet, learning-based methods also yield the need for **sim-to-real systems** and solutions to minimize the gap and provide **interpretability and guarantee** for the generated solutions. Hence, we first developed a framework that can **learn** to communicate between robots based on **Graph Neural Networks** (GNNs) towards better individual decision-making given its local information **in a decentralized manner**. This framework is composed of an encoder (i.e. Convolutional Neural Network) that extracts adequate features from local observations, and a GNN that learns to explicitly communicate these features among robots, and Multilayer Perceptron for action selection. By jointly training these components, the system can learn to determine best what information is relevant for the team as a whole and share this to facilitate efficient path planning. Following up with that, we propose Message Aware Graph Attention neTwork (MAGAT) to combine a GNN with a key-query-like attention mechanism to improve the effectiveness of inter-robot communication. We demonstrate the generalizability of our model by training the model on small problem instances and testing it on increasing robot

density, varying map size, and much larger problem instances (up to $\times 100$ the number of robots).

To port our solution into the real world, we developed a ROS-based system that allows for the fully decentralized execution of GNN-based policies. We demonstrated our framework on a case study that requires tight coordination between robots, and presented first-of-a-kind results that showed successful real-world deployment of GNN-based policies on a decentralized multi-robot system relying on ad-hoc communication. Extending this system, we proposed a vision-only-based learning approach that leverages a GNN to encode and communicate relevant viewpoint information to the mobile robots. During navigation, the robot is guided by a model that we train through imitation learning to approximate optimal motion primitives, thereby predicting the effective cost-to-go (to the target). Our experiments demonstrated its generalizability in guiding robots in previously unseen environments with various sensor layouts.

Vanilla GNN-based decentralized path planning has demonstrated its performance empirically via an end-to-end learning approach. However, these black box approaches are facing challenges to directly deploy in the actual workplace, as they are hard to find a guaranteed and interpretable solution. Therefore, we designed Graph Transformer, as a heuristic function, to accelerate the focal search within Conflict-Based Search (CBS) in a non-grid setting, especially dense graphs. Our framework guarantees both the completeness and bounded suboptimality of the solution. For the explainability and interpretability for RL, we introduced a global path planning algorithm (for example, A*) to generate a globally optimal path, which act as part of the reward function to encourage the robot to explore all potential solutions 'weekly supervised' by the optimal path. As our reward function is independent of the environment, our trained framework generalizes to arbitrary environments and can be used to solve the multi-robot path planning problem in a fully distributed reactive manner.

Throughout my Ph.D. research, I first proposed communication-aware motion planning for multi-robot coordination, where GNNs are introduced to build communication channels for multi-robot teams so that they can learn how to communicate with each other explicitly. The feasibility of this novel research idea has been validated by various simulation experiments based on an end-to-end imitation learning pipeline. To port them into reality, we built a ROS2-based system with adhoc communication to demonstrate our idea in a multi-robot passage scenario and single-robot navigation assisted by randomly sampled camera-based sensors in an unknown environment. Finally, we developed methods that provide interpretation and performance guarantees in the previous black box approaches by introducing a heuristic function into the focal search of CBS and designing a novel reward mechanism called G2RL.

Table of contents

List of figures **vii**

1 Introduction **9**
1.1 Introduction 9
1.2 Achievements 13
1.3 Organization 15

2 Literature Review and Background **17**
2.1 Multi-Robot Motion Planning 17
2.1.1 Reactive Approaches 18
2.1.2 Deliberative Approaches 19
2.1.3 Communication-Aware Approaches 23
2.2 Learning-Based Path Planning 25
2.2.1 Learning-Based Path Planning for a Single Robot 26
2.2.2 Learning-Based Path Planning for Multi-Robot Systems 27

3 Decentralized Multi-Robot Motion Planning **33**
3.1 Introduction 34
3.2 Related Work 35
3.3 Problem formulation 37
3.3.1 Problem. 37
3.3.2 Assumptions. 38
3.3.3 Communications. 38
3.3.4 Preliminaries 39
3.4 Graph Neural Networks for Decentralized Multi-Robot Path Planning . . . 41
3.4.1 Architecture 41
3.4.2 Performance Evaluation 47

3.5 Message-Aware Graph Attention Networks for Large Scale Multi-Robot Path Planning . . . 53
3.5.1 Message-Aware Graph Attention neTwork . . . 53
3.5.2 Properties of MAGAT . . . 54
3.5.3 Architecture . . . 56
3.5.4 Experiments . . . 59
3.6 Conclusion and Future Work . . . 65

4 Real World Deployment of Data-driven Policies for Multi-Robot Motion Planning 67
4.1 Introduction . . . 67
4.2 Related Work . . . 68
4.3 ROS2-based System for Deploying Decentralized GNN-based Policies . . . 71
4.3.1 Preliminaries . . . 72
4.3.2 Approach . . . 73
4.3.3 Network Infrastructure . . . 76
4.3.4 Case Study: Navigation Through A Narrow Passage . . . 76
4.4 Learning to Navigate using Visual Sensor Networks . . . 81
4.4.1 Problem Formulation . . . 82
4.4.2 Visual Navigation using Sensor Networks . . . 83
4.4.3 Zero-Shot Real World Transfer . . . 87
4.4.4 Results . . . 89
4.5 Discussion and Further Work . . . 93

5 Data-driven Heuristic for Path Planning 95
5.1 Introduction . . . 95
5.2 Background and Related Work . . . 96
5.3 Accelerating Multi-Agent Planning using Graph Transformers with Near-Optimal Guarantees . . . 97
5.3.1 Problem Formulation . . . 98
5.3.2 Background: Conflict-Based Search with Biased Heuristics . . . 99
5.3.3 Graph Transformers as Heuristic Functions . . . 102
5.3.4 Experiments . . . 106
5.4 Mobile Robot Path Planning in Dynamic Environments through Globally Guided Reinforcement Learning . . . 108
5.4.1 Problem Description . . . 109
5.4.2 RL-Enhanced Hierarchical Path Planning . . . 111

5.4.3 Implementation 114
5.4.4 Results 117
5.5 Conclusion 122

6 Conclusion and Future Work **123**
6.1 Experts and Data Generation 123
6.2 Communication Strategies for Decentralized Control 124
6.3 Sim-to-Real Transfer 125
6.4 Bounded-suboptimality and Interpretability 127
6.5 Future Avenues 128

References **131**

Appendix A Appendix for Chapter 3 **149**

Appendix B Appendix for Chapter 4 **153**
B.1 Appendix for Learning to Navigate using Visual Sensor Network 153

Appendix C Appendix for Chapter 5 **163**
C.1 Appendix for Mobile Robot Path Planning in Dynamic Environments through Globally Guided Reinforcement Learning 163

List of figures

1.1 Typical application scenarios of multi-robot systems. 9
1.2 Visualization of multi-robot motion planning 10
1.3 Applications of learning to optimization problems. (A) embodies techniques for learning optimization heuristics; (B) embodies techniques for learning to solve POMDPs; (C) is the emerging topic discussed here, embodying techniques for learning to coordinate large systems in real-world applications. 12

2.1 Examples of the scenario that result in incomplete (Fig.2.1a) or sub-optimal solutions (Fig. 2.1b) in prioritized planning. 21

3.1 Illustration of the proposed framework. *(i)* The input tensor is based on a binary map representation (1st channel: partial observation of the environment; 2nd channel: the position of goal ($\mathbf{p}^i_{goal}$), or its projection onto the boundary of the field-of-view; 3rd channel: self (agent) at center, with other agents within its field-of-view). *(ii)* The decentralized framework consists of a CNN to extract observations from the input tensor, a GNN to exchange information between the neighboring agents, and an MLP to predict the actions. *(iii)* Training is performed through cross-entropy loss over a discrete action space. 42
3.2 Illustration of the inference stage: for each robot, the input map $\mathbf{Z}^i_t$ is fed to the trained framework to predict the action; collisions are detected and prevented by collision shielding. The input map $\mathbf{Z}^i_t$ is continuously updated until the robot reaches its goal or exceeds the timeout T_{max}. 45
3.3 Results for success rate (α) and flowtime increase (δ_{FT}), as a function of the number of robots. For each panel, we vary the number of communication hops ($K \in [1,2,3]$), including results obtained through training with the online expert (OE). We also compare our framework with Discrete-ORCA [1][2]. 49

3.4 Success rate and flowtime increase. The rows represent the number of robots on which each model was trained, and columns represent the number of robots at test time. The heatmap maps performance to a color range where purple indicates the best performance and red indicates the worst performance. 50

3.5 Results for success rate and flowtime increase, as a function of the number robots tested on. We vary the GNN implementation ($K \in [2,3]$), trained ('TR10') on a 20×20 map with 10 robots, and GNN implementation ($K = 3$) trained ('TR20') on a 28×28 map with 20 robots. Testing was performed on maps that maintain constant effective robot density. 51

3.6 Histogram of proportion of cases distributed over the number of robots reaching their goal; the network with hop count $K \in [2,3]$, is trained on 10 robots and tested on 60 robots, with and without the OE. 51

3.7 Our proposed decentralized framework. (i) illustrates how we process the partial observations of each robot into input tensor $\mathbf{Z}_t^i$, and how we construct the dynamic communication network. (ii) demonstrates the processing pipeline consisting of a feature extractor, a graph convolution module, and an Multi-layer Perceptron (MLP). The optional skip connection represents the bottleneck structure discussed in Sec. 3.4.1. (iii) visualizes how our model gathers features, computes attention weights by a key-query-like attention mechanism (sandy brown), and selectively aggregates useful features. . . . 56

3.8 The *success rate* (α) and *flowtime increase* (δ_{FT}) against the change of environment setup. Here we present the results of GNN models on the left two columns and MAGAT models on the right; we include `HCA` and `Replan` as baselines. The first row is for "Same Robot Density Set", while the second is for "Increasing Robot Density Set". These figures show the effects of reducing bandwidth or using bottleneck structure. In the legend (`[Graph_Layer_Name]-[Type]-[Num_Features]`), `Graph_Layeer_Name` are `GNN` or `MAGAT`, while `Type` - "`F`" and solid line refer to normal CNN-MLP-GNN/MAGAT-MLP-Action pipeline, and "`B`" and dashed line refer to a bottleneck structure. `[Num_Features]` includes 128 (red), 64 (blue), 32 (black), and 16 (purple). 62

3.9 Generalization test on Large Scale Map Set. a) shows the *success rate*. b) shows the percentage of successful robots ($p_{\text{rg}} = \frac{n_{\text{robots reach goal}}}{n_{\text{robot}}}$), indicating that those model with low *success rates* can still successfully navigated most of the robots to their goals. 65

4.1 The robots form a graph based on their separation and communication range R_{COM}. They leverage communication via latent messages $\mathbf{m}^{i,t}$ generated from local observations $\mathbf{z}^{i,t}$ propagated over graph edges (wireless *Adhoc* communication links) to overcome the partial observability of the workspace. To solve this task, we utilize and deploy Graph Neural Network (GNN)-based policies that aggregate messages of robots within the local neighborhood $\mathscr{N}^{t,i}$ and compute a local action. 72

4.2 Our ROS2 architecture is composed of the *world* and *agents*. There is one agent $i = 1$ in the centralized case, and multiple $i \in \{1 \dots n\}$ in the decentralized case. The agents receive sensor information $\hat{\mathbf{z}}_t$ from either the motion capture system, GPS, or simulator. The aggregator combines sensor information with messages $\mathbf{m}^{t,i}$ to produce observation $\mathbf{z}^{t,i}$ and neighborhood messages $\mathbf{m}^{t,j}$; $j \in \mathscr{N}^{t,i}$, for the policy π_θ^i to generate action $\mathbf{a}^{t,i}$. The control node converts the action into velocity commands $\mathbf{v}^{t,i}$. In *simulation* mode, control drives the simulator instead of the robot wheel motors. The state server orchestrates termination, resets, and operational mode syncs during sequential episodes. This system allows us to run agents in simulation and the *real-world* concurrently, over multiple episodes, and without any human intervention. 74

4.3 The framework configurations used in our experiments. The ROS2 infrastructure is either centralized or decentralized, with varying degrees of decentralization depending on the network setup. We refer to these four configurations as Centralized, Offboard, Onboard over Infrastructure, and Onboard over Adhoc. Observations $\mathbf{z}^{t,1} \dots \mathbf{z}^{t,n}$ feed into centralized policy π_θ or local policies $\pi_\theta^1 \dots \pi_\theta^n$ to produce actions $\mathbf{a}^{t,1} \dots \mathbf{a}^{t,n}$ for agents $1 \dots n$. Local policies consist of a GNN and pass messages $\mathbf{m}^{t,1} \dots \mathbf{m}^{t,n}$ to communicate. In the centralized case, a single policy produces actions for all agents at once in a synchronized manner. For *Offboard*, local policies run asynchronously, exchanging messages over `localhost`. The PC is removed for *Onboard o/Infra*, moving inference onto the robot computers. *Onboard o/Adhoc* is fully decentralized – the agents forgo the router and communicate directly using Adhoc networking. 75

4.4 We deploy a set of five DJI RoboMaster robots in a real-world setup using GNNs and Adhoc communication. The robots navigate through a narrow passageway to reconfigure on the other side, as quickly as possible. 77

4.5 We visualize a variety of makespan and position distributions over the six experiments we conducted. The columns show the data of the centralized simulation baseline in orange and the data of the corresponding real-world experiment as labeled in the column headers in blue. For each experiment, we run a total of 192 episodes with 16 different start and goal positions. The last column compares the Onboard o/Adhoc experiment with a simulation evaluated with communication delays. The first row shows the probability densities of makespans of successful episodes (episodes that did not result in a collision with the wall and for which all robots reached their goal, indicated with N). The median makespan is indicated with a dashed line. The second row shows the distribution of positions, indicating the position of the wall and the passage. The third row shows the distribution of minimum distances between robots at each time step d_{min} and distance from the origin or passage d_{origin}. 79

4.6 Our setup consists of (a) an environment populated with obstacles, visual sensors (blue), a mobile robot equipped with a sensor, and a green target, where the mobile robot has to find the shortest path (red, taken path orange) to an occluded target by incorporating communicated sensor information to enhance its navigation performance. Our framework leverages a setup consisting of a simulation environment (b) that corresponds to the real-world setup (c). The workspace is endowed with custom-built sensors with fish-eye cameras (d, e) that are capable of communicating with each other. One sensor is attached to a mobile robot (e) that acts on the output of that sensor. 82

4.7 We train a policy in simulation using simulated images o_t^i. The CNN $\phi(\cdot)$ encodes images into features z_t^i and the GNN $\theta(\cdot)$ further encodes this for multiple communication hops in L layers as $z_t^{i,l}$. Eventually, the post-processing MLP $\psi(\cdot)$ generates cost-to-go advantages A_t^i which are used to sample a direction to the target along the shortest path as u_t^i and eventually to generate action a_t for the robot, which is equipped with sensor S_1. 85

4.8 Our sim-to-real framework. (a) After training the policy in simulation, we collect image pairs from simulation o^{sim} and the real world o^{real} and use them to train a real-to-sim translator model $\hat{\phi}$ (gray) by reconstructing latent features from the simulation domain z generated through the encoder trained in simulation ϕ (orange) with real-world images. (b) We combine the translator model $\hat{\phi}$ trained on real-world images (gray) with θ and ψ trained in simulation (orange) to deploy the policy to a real-world setup. 88

4.9 We record the performance of the policy trained for $D_S = 2$ and $L = 2$ for a range of different communication ranges and number of sensors. Left: The large environment (blue) is populated with $N = 13$ sensors and the small environment (orange) with $N = 7$ sensors. Right: The large environment (blue) has a communication range of $D_S = 3.5m$ and the small environment (orange) of $D_S = 1.5m$. Both values result in peak performance for the maximum number of sensors in the corresponding environment, as can be seen on the left side. It can be seen that increasing communication range and number of sensors benefits the SPL. 91

4.10 (a) Evaluation of real-world results for three different environments for the fully connected policy. We report success rate and SPL for runs where the target was within line-of-sight (LOS, $M_A = 2$, $M_B = 5$, $M_C = 1$) and outside line-of-sight at the start of the experiment (NLOS, $M_A = 23$, $M_B = 19$, $M_C = 24$). (b, c, d) A selection of two policy evaluations for NLOS configurations for each real-world environment. Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert P and the orange path the path p chosen by the policy π. 92

5.1 **Left:** Examples of our graph-based MAPF instances. To construct the graph, we sample vertices randomly from the free space and connect them with collision-free edges. **Right:** Problem instances that our approach solves while other baselines fail. Different colors represent the trajectories of different agents. Vertices in the same trajectory have deeper colors if their respective time steps are later. 101

5.2 The proposed Graph Transformer architecture. It has several desired properties that are specifically designed to deal with MAPF inputs. See Section 5.3.3 for more details. 102

5.3 The training framework. We use a supervised Contrastive Loss. The labels are generated from the CBS search tree. 104

5.4 Success rates, computation time, and flowtime within the runtime limit of 5 minutes, as functions of the number of agents. The results are averaged over 100 test instances for each setting of the agent number. We evaluate our approach with $w \in [1.005, 1.01, 1.05, 1.1, \infty]$, and compare its performance with CBS, ECBS ($w = 1.1$) and ORCA. Though trained with relatively few agents, results have shown that our approach generalizes well and significantly outperforms the baselines (CBS, ECBS, and ORCA). 104

5.5 We conduct 4 various ablation studies to evaluate the proposed method systematically. See Section 5.3.4 for more details. 106
5.6 The overall structure of our method. The input of each step is the concatenation of the transformed local observation and the global guidance information. A sequence of historical inputs is combined to build the input tensor of the deep neural network, which outputs a proper action for the robot. 111
5.7 An illustration of our reward function. The green, black, yellow and red cells represent the static obstacle, the global guidance, the dynamic obstacle and the robot, respectively. At $t = 7$, the robot reaches a global guidance cell c_i and receives the reward based on the number of eliminated guidance cells. . 112
5.8 Map examples. The black and colored nodes represent the static and dynamic obstacles respectively. Map parameters can be found in Section 5.4.3. . . . 115
5.9 Multi-robot path planning results. The upper row shows the number of reached robots at different steps of different approaches in three testing maps. The solid lines show the average number across 100 tests, and the shadow areas represent the standard deviations. The lower row plots the corresponding histograms of flowtime values. The flowtime of all the failure cases is set to the maximum time step 100. 121

A.1 Two series maps from 28×28, 35×35, 40×40, 45×45 and 50×50 maps as exampples. 149
A.2 Illustration of the inference stage: for each robot, the input map $\mathbf{Z}_t^i$ is fed to the trained framework to predict the action; collisions are detected and prevented by collision shielding. The input map $\mathbf{Z}_t^i$ is continuously updated until the robot reaches its goal or exceeds the timeout T_{max}. 150
A.3 Average frequency of the predicted collisions of individual robots ($\mu = \frac{1}{N_{\text{cases}}} \sum \frac{N^i_{\text{predictedcollision}}}{T^i_{\text{Steps}}}$) for two different testing set. 151

B.1 Generation of training data. (a) We first create random maps; in the illustration, blue points are sensor locations, the red square is the robot, green square is the target and corresponding lines indicate the shortest path from sensors and robots to the target. Black boxes indicate obstacles. (b) Random maps are rendered in 3D in the simulation environment, Webots. Sensors are blue, the target is green, the boxes brown and the robot black. (c) Sensors are equipped with omnidirectional cameras. 157

B.2 Sample of the Twin Environment setup. The simulated environment in Webots can be seen on the left, and the corresponding real-world environment on the right. Note the matching position and alignment of environment size and obstacles. 157
B.3 Samples of the interpreter used to convert latent image encodings into interpretable images. Columns: 8 independent samples. First row: Simulated image o^{sim}. Second row: The image in the first row is encoded as $z^{\text{sim}} = \phi(o^{\text{sim}})$ and reconstructed as $\phi^{-1}(z^{\text{sim}})$. Third row: Corresponding real-world image o^{real}. Fourth row: Reconstruction of the simulated image from the real image as $o^{\text{realtosim}} = \phi^{-1}(\phi_{\text{POST}}(\hat{\phi}(o^{\text{real}})))$. 158
B.4 Generalizability to larger environments and larger number of agents $N = 13$ for GNN layers $L = 2$ and communication range $D_S = 3.5$ m (SPL 0.91). Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π. 158
B.5 All real-world evaluations for Environment A. Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π. 159
B.6 All real-world evaluations for Environment B. Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π. 160
B.7 All real-world evaluations for Environment C. Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π. 161
B.8 A selection of policy evaluations for NLOS configurations in simulation. Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π. 162

C.1 Histograms of the number of eliminated global guidance cells which have not been traveled by the robot across 100 tests in each environment map. $||c_{goal} - c_{start}||_{L1}$ represents the Manhattan distances between the start and goal cells, which are set to 50, 100, and 150. 164

Acknowledgements

Inspired by the Chinese philosopher 'Mozi' and his ideas on 'Universal Love' and 'non-attack', I believe that technology should contribute to humanity by promoting a healthier, safer, and higher quality of life.

I am an enthusiast and a firm believer in the new revolution of robotics, specifically Artificial General Intelligence for Robotics and Embedded Agents. I firmly believe that the concept of 'real2sim (meta-verse) and sim2real' has the potential to bring together the communities of machine learning, computer vision, graphics, and robotics, propelling us into an era of technology beyond traditional first-principle-based methods and control theory. However, there is a significant research gap that needs to be addressed in this revolution. I believe that robotics can offer one of the solutions to the challenges posed by population aging through the development of autonomous systems and human-robot collaboration. This Ph.D. thesis marks the starting point of my contribution to this vibrant and innovative community.

Throughout the journey of completing this Ph.D. thesis, I would like to acknowledge the invaluable support and assistance provided to me. This endeavor would not have been possible without the contributions and guidance of numerous individuals and organizations, to whom I am deeply grateful.

First and foremost, I express my deepest gratitude to my supervisor, Prof Amanda Prorok, for her unwavering dedication, profound knowledge, and insightful guidance. Her expertise, encouragement, and constructive feedback were instrumental in shaping this research and refining my academic abilities. I am truly indebted to her for her patience, mentorship, and the countless hours they dedicated to my intellectual growth.

I would also like to extend my sincere appreciation to the members of my supervisory committee, Prof Guillaume Adrien Sartoretti and Prof Thomas Sauerwald, for their valuable feedback, suggestions, and rigorous examination of my thesis. Their expertise and scholarly contributions greatly enhanced the quality of this work, and I am grateful for their time and expertise.

I am grateful to the University of Cambridge for providing me with the necessary resources, facilities, and a stimulating academic environment to pursue my Ph.D. research.

The support and encouragement from the faculty members, administrative staff (Lise Gough, Joy Rook and Marketa Green), and my fellow colleagues have been invaluable in shaping my research experience.

I would like to express my gratitude to my parents (Guojun Li and Cuizhao Ye) and sister (Limin Li), and grandma (Julan Liang) for their unwavering support, understanding, and patience throughout this demanding journey. Their encouragement, love, and belief in my abilities have been a constant source of motivation, and I am forever grateful for their presence in my life.

Lastly, I wish to acknowledge all the authors, researchers, and scholars (Dr Fernando Gama, Prof Alejandro Ribeiro, Dr Alex Raymond, Dr Nikhil Churamani, Jan Blumenkamp, Weizhe Lin, Prof Zhe Liu, and Binyu Wang) whose work has been a source of inspiration and guidance for my research. Their contributions to the field have laid the foundation for my study, and I am indebted to their dedication and commitment to advancing knowledge.

Although it is impossible to name everyone who has contributed to my Ph.D. journey, I am sincerely thankful to each and every individual who has played a role, no matter how big or small. Your support, encouragement, and belief in my abilities have been invaluable, and I am truly honored to have had the opportunity to learn from and work alongside such remarkable individuals.

Thank you all for being a part of this significant milestone in my academic and personal life.

Chapter 1

Introduction

1.1 Introduction


(a) Mobility-on-demand [3]


(b) Connected vehicles [4]


(c) Automated warehouse [5]

Fig. 1.1 Typical application scenarios of multi-robot systems.

For the first time in human history, over half of the world's population resides in cities, and that figure is set to rise to two-thirds by 2050 [6]. This ongoing urbanisation is burdening critical infrastructure and logistics systems. In order to relieve this pressure, we must look towards systems that streamline and optimise operations.

The field of robotics research is posed to respond to these challenges by creating a new era with precision agriculture, fast manufacturing processes, and smart cities by utilising various kinds of robots and intelligent vehicles. The use of industrial robots and domestic mobile robots has increased, gradually bringing the concept of robotics into everyday life. These singleton robots have limited abilities to handle large-scale tasks, whereas cooperative robot systems comprised of multiple robots can handle more complex missions. Examples of this include the use of multi-robot systems for mobility-on-demand [3], connected vehicles [4], and automated warehouses [7] (Fig. 1.1).

Research on multi-robot systems considers methods to control multiple robots to accomplish a common objective in dynamic environments, as well as methods that enable robots to

understand and interpret their surrounding world [8]. A robot team would undertake a wide range of tasks, from fundamental behaviours, such as navigation, movement and communications to complex behaviours like foraging and playing football [9]. However, multi-robot systems face challenges in scalability and computational complexity. This challenge is compounded by faults in the robot teams. The essential underlying function of controlling multi-robot systems is to navigate individual robots to their target locations without collisions. This field of research is called multi-robot motion planning. As illustrated in Fig. 1.2, there is a heterogeneous robot team consisting of multiple ground robots (r_i, for all $i = 1,\ldots,6$) and aerial robots (r_i, for all $i = 7,8,9$). The ground robots need to avoid static obstacles (trees, stones, building, lake and desert) and other robots, while the aerial robots can fly over lakes (i.e. for robot r_7) and desert (i.e. for robot r_8), while high trees and buildings will affect the mobility of the drones (i.e. for robots r_8 and r_9). Therefore, collision-free paths are generated to guide the ground robots (r_i, for all $i = i = 1,\ldots,6$) to their goals (g_i, for all $i = 1,\ldots,6$, dash line), and the aerial robots (r_i, for all $i = 7,8,9$) to their destinations (g_i, for all $i = 7,8,9$, solid line). However, the increasing number of robots and the heterogeneity of the team expands the search space of the states of robots drastically, which makes it very hard to generate collision-free trajectories optimally and quickly for such heterogeneous teams of robots.

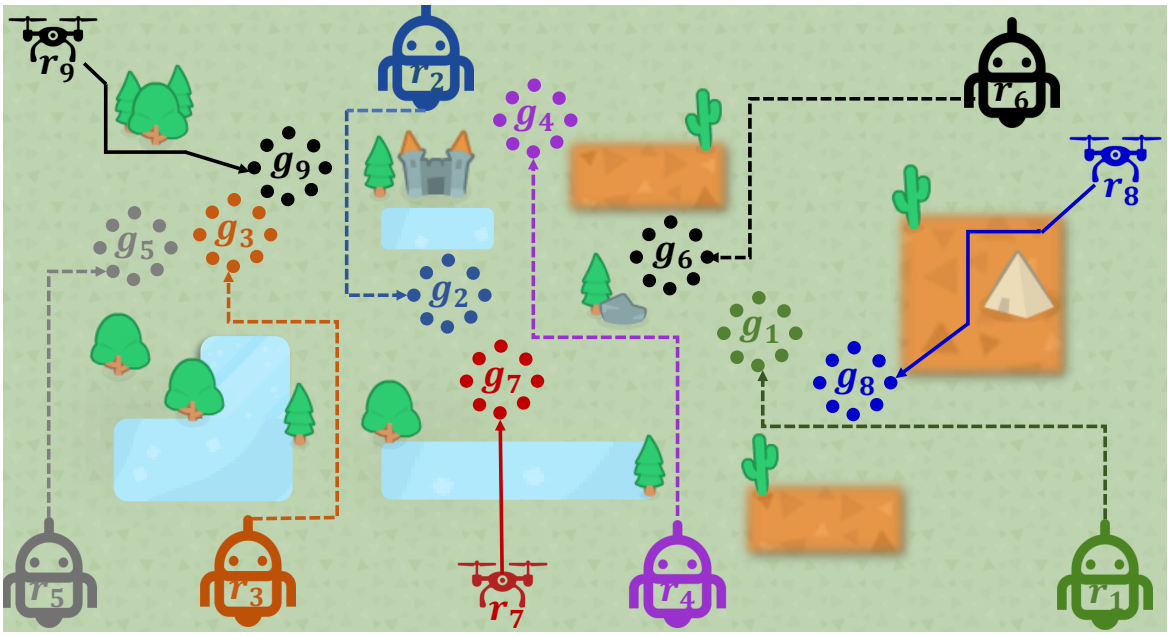


Fig. 1.2 Visualization of multi-robot motion planning

There is a trade-off between the optimality, completeness and computational complexity of the solution. For instance, the coupled approach considers all states of the robot as a

composite so that it can find an optimal and complete solution. The required computation grows exponentially as the number of robots increases. In decoupled approaches, the independent paths will be generated for each robot, and then resolve conflicts between these paths with specialised methods. This method sometimes cannot find an optimal solution and/or guarantee completeness.

Efficient and collision-free navigation in multi-robot systems is fundamental to advancing mobility. The problem, generally referred to as Multi-Robot Path Planning (MRPP) or Multi-Agent Path Finding (MAPF), aims at generating collision-free paths leading robots from their origins to designated destinations. Current approaches can be classified as either *coupled* or *decoupled*, depending on the structure of the state space that is searched. While coupled approaches are able to ensure the optimality and completeness of the solution, they involve *centralized* components, and tend to scale poorly with the number of robots [10, 11]. Decoupled approaches, on the other hand, compute trajectories for each robot separately, and re-plan only in case of conflicts [1, 12, 13]. This can significantly reduce the computational complexity of the planning task, but generally produces sub-optimal and incomplete solutions. Balancing optimality and completeness with the complexity of computing a solution, however, is still an open research problem [14, 15].

Learning-based methods have proven effective at designing robot control policies for an increasing number of tasks [16, 17]. The application of learning-based methods to multi-robot planning has attracted particular attention due to their capability of handling high-dimensional joint state-space representations, by **offloading the online computational burden to an offline learning procedure** [18, 19]. We argue that these developments point to a fundamental approach that combines ideas around the application of learning to optimization and produce a flexible framework that could tackle many hard but important problems in robotics, including multi-agent path planning [20], area coverage [21, 22], task allocation [23–25], formation control [26], and target-tracking [27]. In this thesis, we motivate this approach and discuss the crucial challenges and research questions.

The concepts presented here are part of a bigger picture of the use of learning to the solving of optimization problems. In Fig 1.3, we consider how learning is applied to either increase the scale of solvable problems or to increase the ability to deal with practical, partial-information problems. Along the problem scale axis, for example, the operations research community has made use of learned heuristics to solve TSPs [28, 29], VRPs [30], and general MILPs [31]. Along the information axis, which includes dealing with POMDPs, techniques such as RL play a major role, as well as ideas such as tuning Monte-Carlo Tree Search [32], embedding learned components into optimal control frameworks [33], and learning how to bias sampling planners [34].

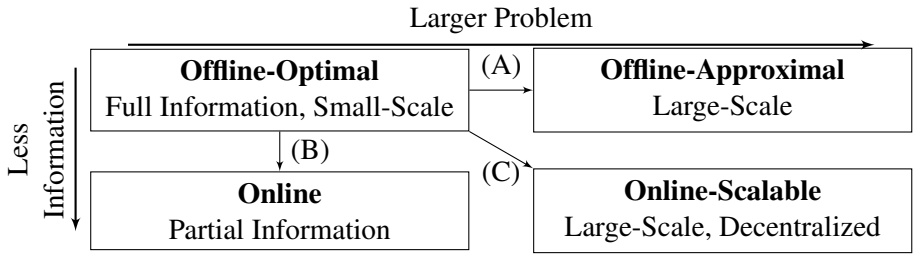


Fig. 1.3 Applications of learning to optimization problems. (A) embodies techniques for learning optimization heuristics; (B) embodies techniques for learning to solve POMDPs; (C) is the emerging topic discussed here, embodying techniques for learning to coordinate large systems in real-world applications.

Practical multi-robot planning and control build on the progress along both of these axes: the degrees of freedom and environment complexity increase, while the ability to communicate and coordinate at scale decreases. Traditional ***centralized*** approaches would use a planning unit to produce coordinated plans that agents use for real-time on-board control; these have the advantage of producing optimal and complete plans in the joint configuration space but true optimality is NP-hard in many cases [20] and they will struggle when communications are degraded and frequent replanning is required. By contrast, ***decentralized*** approaches reduce the computational overhead [35] and relax the dependence on centralized units [1, 2] to deal with challenged communications, but account for local objectives and cannot explicitly optimize global objectives (e.g., path efficiency).

What the directions of Fig 1.3 teach us is that success follows from starting with simple problems and using their examples to approach complex ones. This progression from example to the application is reminiscent of *Imitation Learning*, and we use this crucial observation to understand how learning can play a role in mitigating the shortcomings of decentralized approaches in solving challenging multi-robot problems.

Bridging the gap between the qualities of centralized and decentralized approaches, learning-based methods promise to find solutions that *balance optimality and real-world efficiency*. The process of generating data-driven solutions for multi-robot systems, however, cannot directly borrow from single-robot learning methods because *(a)* hidden (unobservable) information about other robots must be incorporated through learned communication strategies, and *(b)*, although policies are executed locally, the ensuing actions should lead to plans with a performance near to that of coupled systems. Vanilla machine-learning-based path planning has demonstrated its performance empirically via an end-to-end learning approach. However, these black box approaches are facing challenges to directly deploy in the actual workplace, as these methods find it hard to guarantee a guaranteed and interpretable solution. This agenda means that we need to address the following content:

- how to generate multi-robot training data;
- how to generate decentralizable policies;
- how to transfer these policies to real-world systems;
- how to generate a bounded-optimality solution or interpret the solution.

The following section elaborates these four key challenges and indicates promising directions.

Though planning complexity is reduced with a decentralized approach, the use of a learning-based approach requires consideration of state-action space coverage, especially since introducing multiple agents causes the exponential growth of the joint state-action again. This core challenge is the reason why the development of learning-based multi-robot controllers is a nascent field. While a number of learning paradigms have been applied to this topic (e.g., RL [36, 37]), this direction of research initially focuses on imitation learning strategies, and then extends into reinforcement learning approaches.

This research direction is motivated by developing machine learning solutions for multi-robot motion planning, computing **near-optimal** solutions using **minimal online computation**. A decentralized planner with multiple communication channels is proposed to **offload the nominal online computation of coupled optimal plans to an offline learning procedure**. This series of work can be trained and self-improved until it is able to assign and guide a large number of robots to their destination stably, optimally, and fast, which has not been achieved before [38].

1.2 Achievements

We want to emphasize that the key contribution of this research includes the novel idea of using Graph Neural Networks to build a communication channel between agents towards decentralized multi-agent path planning framework [19], the capability of generalization in this framework using attention mechanism [39]. This novel framework can also be deployed into a different domain, including Sub-modular Action Selection for object tracking [40] and navigation based on sensor network [41]. We then demonstrated how this GNN-based policy can be used to navigate a robot team through a narrow passage in the real world [42].

I am the leading author of [19] and [39], which are also the unique contributions of my doctoral research. I also shared the work in knowledge in GNN, conducting the physical experiment and intellectual leadership in collaborative work within ProrokLab [2, 41, 42] and KumarLab [40].

The work developed for this thesis resulted in three conferences, and one journal publication, listed below. For those published works that are not present as main chapters in this thesis, I briefly review them as related work in corresponding sections.

- Qingbiao Li, Fernando Gama, Alejandro Ribeiro, Amanda Prorok. "Graph Neural Networks for Decentralized Multi-robot Path Planning," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020.
- Qingbiao Li, Fernando Gama, Alejandro Ribeiro, Amanda Prorok. "Graph Neural Networks for Decentralized Path Planning," International Conference on Autonomous Agents and MultiAgent Systems (AAMAS), Extended Abstract, 2020.
- Qingbiao Li, Weizhe Lin, Zhe Liu, Amanda Prorok. "Message-Aware Graph Attention Networks for Large-Scale Multi-Robot Path Planning," IEEE Robotics and Automation Letters (RA-L). 2020.
- Chenning Yu*, Qingbiao Li*, Sicun Gao, Amanda Prorok, Accelerating Multi-Agent Planning using Graph Transformers with Near-Optimal Guarantees, International Conference on Robotics and Automation (ICRA), 2023. (Joint first author)

Some claims presented in this thesis are also supported by the following studies:

- Jan Blumenkamp, Qingbiao Li, Binyu Wang, Zhe Liu and Amanda Prorok. See What the Robot Can't See: Learning Cooperative Perception for Visual Navigation. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2023.
- Fernando Gama, Qingbiao Li, Ekaterina Tolstaya, Amanda Prorok, Alejandro Ribeiro. "Synthesizing Decentralized Controllers with Graph Neural Networks and Imitation Learning," IEEE Transactions on Signal Processing (TSP), 2022.
- Binyu Wang, Zhe Liu, Qingbiao Li, Amanda Prorok. "Mobile Robot Path Planning in Dynamic Environments through Globally Guided Reinforcement Learning," IEEE Robotics and Automation Letters (RA-L). pp. 6932–6939, 2020.
- Jan Blumenkamp, Qingbiao Li, Amanda Prorok. "Evaluating the Sim-to-Real Gap of Graph Neural Network Policies for Multi-Robot Coordination," IEEE International Conference on Robotics and Automation (ICRA), Real World Swarms Workshop, 2021.
- Jan Blumenkamp, Steven Morad, Jennifer Gielis, Qingbiao Li, Amanda Prorok. "A Framework for Real-World Multi-Robot Systems Running Decentralized GNN-Based Policies," IEEE International Conference on Robotics and Automation (ICRA), 2021.

- Amanda Prorok, Jan Blumenkamp, Qingbiao Li, Ryan Kortvelesy, Zhe Liu, Ethan Stump The holy grail of multi-robot planning: Learning to generate online-scalable solutions from offline-optimal experts. International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Blue-sky, 2022.

1.3 Organization

This thesis is organized as follows. State of the art in conventional-based and learning-based path planning are reviewed in Chapter 2. Chapter 3 presents our decentralized framework in multi-robot path planning in a homogeneous team and its extension that considers communication constraints. In Chapter 4, we conduct physical experiments to demonstrate how we deploy GNN-based policy in the real world. Chapter 5 introduces two data-driven heuristic approaches for multi-agent path planning towards interpretable and bounded guaranteed solutions. Finally, Chapter 6 concludes the thesis.

Chapter 2

Literature Review and Background

The aim of research on multi-robot motion planning is to ensure that every robot in a shared area can navigate to its destination using a collision-free trajectory or path, even in environments with static or dynamic obstacles and other robots present. To achieve this, the current methodologies can be generally classified as either reactive or deliberative approaches. Therefore, Section 2.1 in this chapter, will first review the state-of-the-art methods in multi-agent coordination, and discuss the strengths and limitations among different classifications.

Recently, the breakthroughs in utilizing the computational power of GPUs have accelerated the training process of Convolutional Neural Networks (CNNs). These breakthroughs have also attracted the attention of robotics scientists to apply them for path planning based on imitation learning and reinforcement learning. Section 2.2 presents several of the most recent studies in learning-based single agent path planning. It also discusses the performance of Multi-Agent Reinforcement Learning (MARL) for multi-player games and the real world. Recent studies in Graph Neural Networks (GNNs) present their potential in handling data with rich relational information among elements as a graph. Preliminary work has been carried out by using it to represent the communication between agents for multi-agent path planning.

2.1 Multi-Robot Motion Planning

In this section, the state of the art methods in multi-agent coordination are classified into table 2.1 and are discussed with respect to their strengths and limitations.

Table 2.1 Summary of main multi-robot motion planning approaches.

	Reactive techniques	Coupled methods	Decoupled methods
Guaranteed solution	No, deadlocks may occur	Yes	No
Complexity	P; Demonstrated on thousands of agents	NP-hard	P; Demonstrated on tens of robots
Solution quality	Suitable in open space; poor in clutter	Optimal	Optimality and completeness not guaranteed;
Decentralization	Suited, needs to observe states of neighbor robots	Not suited	Suited, requires communication

2.1.1 Reactive Approaches

In reactive approaches, each robot generates an optimal path individually to its goal without considering future interactions with other robots. Following a planned path, each robot monitors the positions and velocities of the neighbouring robots. If there is a potential future collision, the robot tries to resolve it by adjusting its immediate heading and current velocity vector. These methods are computationally efficient but susceptible to deadlocks. Deadlock is a scenario where two or more robots attempt to compute the collision avoiding velocity but result in a zero-velocity vector when they meet within their collision detection distance. The robot will stop without any further motion.

Many collision-avoiding velocity strategies have been proposed within the last three decades. Lumelsky and Harinarayan et al. [43] firstly proposed an algorithm based on maze searching called the cocktail party model, which is analogous to human behaviour in a crowded place: when a guest decides to talk with someone, he will follow this collision-free path towards the person without communicating with others about this intention by assuming others act well. If someone steps in, he will try to avoid this interruption by increasing the safety margin **distance** when passing this person. More recently, techniques based on the **velocity** obstacle paradigm have become dominant [1, 44, 45]. One of the popular methods in this approach is the optimal reciprocal collision avoidance (ORCA) formulation proposed by Van Den Berg et al. [45]. In this method, each robot can observe the velocity of other robots to avoid collisions with them, where the robot needs to select its own velocity from its velocity space in which certain regions are marked as 'forbidden' due to the presence of other robots. In their formulation, each robot owns a half-plane (in velocity-space) of velocities that are allowed to be selected to guarantee collision avoidance. The robot then

selects its optimal velocity from the intersection of all permitted half-planes, which can be done efficiently using linear programming.

Approaches based on reciprocal velocity obstacles have been widely applied in real-world tasks [46] because of their computational efficiency — a collision-avoiding velocity for a robot or other type of agent can be computed in a fraction of a millisecond. Hence, these methods can be used in systems with a large number of agents, including robotic swarms, and crowd simulations in computer games. However, these reactive techniques are significantly constrained due to their reactive-based nature. This causes such planners to be short-sighted in time without considering future states of neighboring agents, and result in undesirable or ineffective behaviour such as deadlock.

2.1.2 Deliberative Approaches

The deliberative approaches are the generalisation of a more widely studied solution for single-robot motion planning, where a sequence of valid configurations is generated for all robots before execution in a given environment. The team of robots will follow their planned path from their start locations to their respective destinations without collision. A collision is defined by a situation where any two agents are at the same location at the same time, or a collision with obstacles in the environments.

In terms of the structure of the searched state space, they can be mainly classified into coupled and decoupled approaches:

- ***Coupled approaches***: are analogous to single-robot motion planning, where the multi-robot systems are considered as one composite robot with many degrees of freedom, and generates a solution by planning a path in a joint configuration space of all robots. Generally, the required computation time increases exponentially with the number of robots;

- ***Decoupled approaches***: compute plan for each robot independently, and resolve conflicts between the paths of individual robots with specialised algorithms. This approach can be computationally more efficient, but sometimes cannot provide a solution. For example, the corridor swap scenario is described in the following section.

Multi-robot motion planning is known to be computationally intractable [47]. The main concerns of this field of research are computation efficiency, optimality, and completeness. The quality of the planning can be measured by **makespan** and **flowtime**. **Makespan** is the maximum length of plans, which is calculated by the distance between the start time of the first agent and the completion time of the last agent. **Flowtime** is the sum of the travel times

of all robots, which is related to the energy consumption of the team. To find an optimal solution that minimizes either makespan and flowtime is a **NP-hard** problem.

Coupled planning

Coupled approaches are the composition of single-robot path planning into a given multi-robot path planning in a joint configuration space, which is constructed by a Cartesian product of configuration spaces of individual robots. The search-based approach has been applied to path planning in the joint configuration space [10, 11, 48]. Firstly, the joint configuration space is discretized in the form of a graph, where vertices represent joint configurations, and the edges represent joint moves that move robots from one joint configuration to another. Search for the minimum cost path is carried out subsequently on this graph. Hence, coupled approaches can find a joint path, which is optimal with respect to the given discretization.

However, coupled approaches have significant limitations in their inherent poor scalability: if we assume a fixed graph discretization of the configuration space of a single robot, the number of vertices in the discretization of the joint configuration space grows exponentially with the number of robots. The worst-case time complexity of minimum-cost path search in a graph is quadratic in the number of vertices, which brings difficulties to find the solution based on a naive search in the joint configuration space.

To overcome this limitation, Standley et al. [49] proposed an idea named operator decomposition. A customised discretisation strategy is applied in modelling the possible combination of the actions of robots based on a tree of single robot motion, where at each vertex only a single robot assigns a move to a joint move. Hence, the search becomes more focused in the joint configuration space, because the heuristic estimate can be evaluated at each state of the tree without exploring every combination of robots' motion.

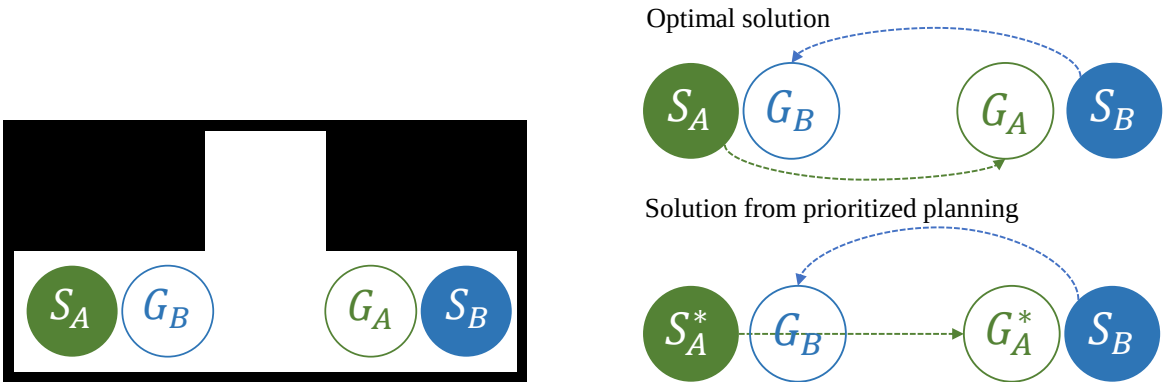
In addition, Wagner and Choset [48] proposed another method that brought significant improvement of the scalability of search in the joint configuration space, which is called independence detection. By this method, the multi-robot path planning problem can be divided into several sub-problems of multi-robot path planning with fewer robots, under the constraints that there is no collision between the optimal solution of these sub-problems. By identifying and solving these independent sub-problems separately, the required dimension of the joint configuration space to search can be significantly reduced and still guarantee the optimality of the solution. However, this method mainly works well in sparse multi-robot path planning problems, where sub-problems can be easily identified without much interaction with others. For dense multi-robot path planning problems, the solution can only be found by original high-dimensional joint search.

Wagner and Choset [50] also proposed the M* algorithm based on sub-dimensional expansion to plan in the joint configuration space. The search will start from a low-dimensional subspace representing configuration spaces of individual robots and increases the dimensionality when the need for coordination is detected. Then Ferner et al. [51] extended M* algorithm with operator decomposition method for an optimal solution, called ODrM*.

Coupled approaches have constraints to implement in a decentralised way, which might affect robustness, execution speed and privacy. There are some proposed methods that can be implemented in a distributed manner, privacy-preserving state space search algorithms called MAD-A* proposed by Nissim and Brafman [52]. In extreme cases, these algorithms may require that each expanded state can be passed through one message. It is impractical because the communication bandwidth of current distributed robotic systems is insufficient to handle large data, especially the expanding search state as the number of robots increases.

Decoupled planning

Decoupled approaches do not search in the joint configuration space, but instead, perform a sequence of path or trajectory planning queries in the configuration space of each robot. These techniques offer better scalability in the number of robots and computational efficiency.



(a) Corridor swap scenario (b) Heads on scenario

Fig. 2.1 Examples of the scenario that result in incomplete (Fig.2.1a) or sub-optimal solutions (Fig. 2.1b) in prioritized planning.

Prioritised planning [12] is one of the typical examples of the decoupled scheme for multi-robot path planning. In prioritized planning, each robot is assigned a unique priority and the algorithm proceeds sequentially from the highest priority robot to the lowest-priority one. In each iteration, one of the robots plans its trajectory such that it avoids collisions with all the higher-priority robots that follow the trajectories planned in previous iterations.

However, there are several limitations to this method. Firstly, this method is likely to work well in uncluttered environments, while it is difficult to find a complete solution if the configuration of obstacles and start and goal are arbitrary for each robot. For example, in the corridor swap scenario (Fig. 2.1a), robot A travels from S_A to G_A, and robot B travels from S_B to G_B in a corridor that is only slightly wider than a body of a single robot. Both robots can travel at identical maximum speeds. In this case, no matter which robot starts planning first, its trajectory will be in conflict with all goal-achieving trajectories of the robot that plans second.

Another issue that arises when decoupled methods are used is the suboptimality of the generated solutions. Clearly, the quality of the generated solutions is highly sensitive to the choice of priorities assigned to the robots, and some researchers investigate techniques for choosing good priorities for the robots. For example, in the heads-on scenario (Fig. 2.1b), two robots are assigned to move from S_A to G_A and S_B to G_B, respectively. In the optimal solution (top picture), each robot can adjust its trajectory to split the total cost of collision-free trajectory equally. However, in prioritized planning, robot A is set to have higher priority than robot B. Therefore, robot A will move along a straight line, regarding the trajectory of robot B, which results in a higher total cost of trajectory. To optimize the makespan (the longest travel time is minimized), Van den Berg [12] proposes a heuristic that assigns a higher priority to the robots with longer travel distances. Additionally, Bennewitz et al. [53] used randomized hill-climbing to find a good priority sequence. Even with these improvements, the optimal priority sequence may still result in an unsatisfactory result, especially when the number of robots is large.

Different from the coupled methodologies, decoupled approaches can be easier to implement in a decentralized fashion. A decentralized prioritized planning technique proposed by Velagapudi et al. [54] allows robots in a team to plan their paths in parallel and replan if a conflict between two paths is detected. This technique can utilize the distributed processing power onboard to generate individual paths in parallel, which can reduce the total computation time to find the solution.

Recently, there has been a trend to explore potential solutions that lie between coupled and decoupled approaches. These approaches allow the various robot-robot behaviors to be achieved using decoupled planning methods while avoiding planning in the large joint configuration space. The Conflict-Based Search (CBS) algorithm proposed by Sharon et al. [55] utilizes a two-level algorithm to find an optimal solution based on a constraint tree, where each node consists of a set of constraints. The conflict is defined by two or more agents that occupy a particular vertex in the graph at the same time. At a high level, a search is performed on a tree based on conflicts between agents. At a low level, a search is performed

only for a single agent at a time. In many cases, this reformulation enables CBS to examine fewer states than A* while still maintaining optimality. This algorithm can obtain an optimal and complete solution and outperform A* in environments with more bottlenecks.

To mitigate the worst-case performance of CBS, Sharon et al. [56] generalised CBS into Meta-Agent CBS (MA-CBS). MA-CBS employs a threshold for merging conflicting agents into a meta-agent, which is then treated as a joint-composite agent by the low-level solver. This framework for multi-agent motion planning is only activated if the number of conflicts exceeds the specified threshold. In the low-level search, any completed multi-agent planner can be utilized, rendering MA-CBS a flexible and adaptable solution for multi-agent motion planning. The above CBS variants only consider the cost of the paths with potential conflicts in the nodes of the CBS constraints tree as the costs of the nodes. Hence, Felner et al. [57] extended this by calculating the admissible heuristics, which adds the h-values to the costs of these nodes.

Li et al proposed an anytime algorithm called MAPF-LNS [58] that combines the strengths of both worlds: anytime algorithms that quickly find an initial solution using efficient existing MAPF algorithms, even for large problems, and that subsequently improve the solution quality to near-optimal as time progresses by replanning subgroups of agents using Large Neighborhood Search. This algorithm can be understood as a near-optimal algorithm (with no guarantees), which combines the strengths of leading algorithms from across the algorithmic spectrum in the sense that it computes initial solutions fast, find near-optimal solutions eventually and scales to very large numbers of agents. Extended from this work, they developed a suboptimal algorithm MAPF-LNS2 [59] based on a large neighborhood search for solving Multi-Agent Path Finding efficiently. An efficient single-agent pathfinding algorithm SIPPS based on SIPP is introduced to find a short path that avoids collisions with a given set of paths and minimizes the number of collisions with another given set of paths. Their comparison with state-of-the-art MAPF algorithms demonstrated that MAPF-LNS2 is fast, scalable, and memory-efficient.

2.1.3 Communication-Aware Approaches

Effective communication within a robot team is crucial for decision-making during robot movements. It enables robots to coordinate their actions, synchronize their movements, and collaborate seamlessly toward a shared objective. By communicating, each robot can share its current state and intended actions with other team members, fostering a cohesive and informed decision-making process. Unfortunately, robust and continuous communication cannot be guaranteed due to bandwidth limitations, or interference from the surroundings. These limitations ultimately affect the optimality of solutions found and the overall resilience of the

team to disruptions. Redundant communication may also burden the computational capacity and adversely affect overall team performance. Hence, new trends of research have been carried out in ***communication-aware path planning approaches*** by explicitly considering communication during path generation, and optimizing this overreaching goal [60–63]: to generate collision-free trajectories for multi-robot teams to reach their destinations.

Generally, the communication between robots in multi-robot systems can be obtained by communicating via an intermediate unit or base station and communicating with individual robots directly. Hence, communication-aware approaches have been explored in both cases. Rahman et al. [61] introduced an algorithm designed to address the positioning of relay robots within a multi-robot mission, with the aim of establishing or improving communication between a static operator and multiple remote units in an obstacle-filled environment. To accomplish this, they employed a layered graph known as a Communication map data structure, which was constructed to encompass the potential relay positions as the unit moved. By computing the communication map graph, it could be reused, thereby reducing the computational load for re-planning. Additionally, they expanded a mini-arborescence tree to establish connections between the operator, relays, and unit, minimizing communication costs. The findings demonstrated that their approach optimally maintained communication quality and exhibited robustness against unexpected malfunctions in the intermediate unit. Caccamo et al. [60] developed the Resilient Communication-Aware Motion Planner (RCAMP), which can be integrated with a robust and online radio signal mapping method based on Gaussian Random Fields. Based on the available sensory information, both the environment and the physical constraint of the robot are under the consideration of RCAMP. If there is a communication loss, a self-repair strategy based on RCAMP can guide the robot into a connection-safe position by considering the signal strength of the surrounding and goal position. Their simulation results show that the robot can reach the goal along the path with optimal communication quality avoiding signal strength drops.

Communications between robots is critical, particularly in decentralized approaches where problems regarding what information should be delivered to whom and when are raised. Fowler et al. [63] introduced a framework called Intelligent Knowledge Distribution to determine the information, content computed by Kullback-Leibler Divergence. With this framework, each agent can determine what information needs to be shared with whom at an appropriate time. This can contribute to the optimal communication schedule, efficient task arrangement and behaviour. Similarly, Best et al. [62] develop a scheduled strategy to communicate valuable information between robots, where robots only start to share information directly when necessary, and only to specific robots that need it. The communication is requested based on the impact of the robot state uncertainty in the performance of the team.

Overall, their method achieves comparable performance to methods that use continuous communication. However, the threshold of when to communicate is manually tuned, and the selection of the right agent to share information with defined through an oversimplified heuristic. To overcome the aforementioned limitations, methods based on imitation learning and reinforcement learning have been applied to multi-agent problems to learn communication protocols without prior definition to solve sophisticated tasks, which will be further discussed in Section 2.2.2.

2.2 Learning-Based Path Planning

Recent breakthroughs have taken place in utilizing the computational power of GPUs to accelerate the training process of convolutional neural networks. In 2012, Krizhevsky et al. [64] developed a large and deep convolutional neural network, called AlexNet, to learn the appearance of numerous and complex objects. The training time is decreased by using Rectified Linear Units (ReLU) instead of using the conventional hyperbolic tangent function. A regularisation method, called Dropout layers, was implemented to reduce overfitting. Additionally, AlexNet was trained on GPUs, which further reduces the training time and increases the acceptable dataset and scale of the images. These methods were widely adopted in various applications of deep learning techniques.

This breakthrough is also promoting increasing activate research in the application of deep learning in the field of robotics, including object detection for an automatic car, imitation learning, and reinforcement learning for motion planning in manipulation. Notably, this trend is also giving rise to exploring learning-based methods for handling path-planning problems for single-agent and multi-agent teams. These learning-based methods have the potential to offload the online computation (high dimensional joint space) to an offline learning procedure effectively.

Multi-robot motion planning is a generalization of a more widely studied problem of single-robot motion planning. Moreover, many of the algorithms for multi-robot motion coordination, solve single-robot motion planning problems as a subroutine. Therefore, we will start our explanation in Section 2.2.1 by reviewing the state of the art in the learning-based method for single-robot motion planning by discussing recent work in search-based and sampling-based motion planning. Section 2.2.1 will review learning-based methods for multi-agent cases, where we will mainly talk about Multi-Agent Reinforcement Learning (MARL).

2.2.1 Learning-Based Path Planning for a Single Robot

Among the various kinds of motion planning methods, search-based motion planning is one the most comprehensive frameworks for reasoning, where a search tree of feasible motion is expanded from the start to its goal [65]. Heuristics are introduced to guide the search towards potentially good directions and consequently minimize search effort, where A* is one of the most popular algorithms. However, the current search-based methods do not explore enough the effect of the heuristic in minimizing the search procedure. Hence, Mohak Bhardwaj et al. [66] proposed a learning-based method where a heuristic policy can be trained to decide wisely which node of the search tree to expand via imitation learning to minimize search effort. Based on this method, the heuristic for search should evolve during the search progress, where the search policy will actively interact with the valid configuration in the space, and focus on searching the region with a higher probability towards its goal without collision. Additionally, this heuristic is a data-driven policy, which can adapt to the distribution of data changes. Imitation learning is introduced to train this sequential decision-making policy, which can utilize the information extracted from an expert trajectory to decide the node to expand next at the current interaction, and also influence the expansion of the wavefront in the next iteration in turn. To achieve a faster convergence rate, full information, called clairvoyant planner, can be observed by the heuristic, which uses dynamic programming (Dijkstra) to efficiently compute the optimal decisions for any given wavefront. Their approach can pave the way forward for learning heuristics that demonstrate an anytime nature, which finds feasible solutions quickly and incrementally refining them over time.

Sampling-based search is also a popular approach for motion planning. Random-exploring Random Trees (RRT) [67] is one of the most well-known methods in this approach, where a tree is expanded iteratively from the start towards the goal by randomly selecting the intermediate points. This approach can handle the constraints of non-holonomic robots, including dynamics and motion planning with a high degree of freedom. However, the random nature of this method cannot guarantee the optimality and completeness of the solutions. For example, a different path with various lengths can be generated in the given same configuration. Therefore, a substantially improved RRT called Neural EXploration-Exploitation Tree (NEXT) was proposed by Chen et al. [68] to train the neural network based experiences the dependency between tasks structures and to reduce the sample requirement for growing a random tree for agent from start to reach the goal. The neural architecture is developed to extract the latent embedding of the task within its configuration space for a customised planning module. Then, this trained network can be integrated with Upper Confidence Bound (UCB)-type algorithm to obtain a balance online between exploration and exploitation in a new task. These two components can improve the performance of both alternatively, where

the UCB-type algorithm will be guided by the learned neural network, while the network will be evolved through the successful path generated by the UCB-type algorithm. Their results show that NEXT can generate the path with very small search tree, which also outperforms previous methods, including RRT* [69], BIT* [70] and CAVE-plan [71].

Besides, Bency et al. [72] proposed a fast and optimal motion planner by using the Long-Short Term Memory (LSTM), to mimic the stepwise output of an oracle planner. The Oracle is the expert algorithm that can always generate optimal paths in a given configuration. The network is trained by L2 norm between predicted output and the oracle, where A* is introduced to be the oracle algorithm for generating the optimal path in a given configuration. The proposed framework was validated in a 2D grid world as well as multi-link robot arm with faster computation and comparable optimality with A* and RRT* [69].

2.2.2 Learning-Based Path Planning for Multi-Robot Systems

Among the family of learning-based methodologies, imitation learning (or supervised learning) and reinforcement learning are the most popular approaches that have been applied in the field of robotics.

In imitation learning, the policy of agent is trained to make a sequential decision in an environment, where the training signal comes from demonstrations of an expert (Oracle) [73]. This method has been widely applied in the scenario, where the tasks are trivial for human expert, but very complex and challenging for a machine due to high dimensional search space, i.e. robot manipulation with the high-degree freedom robotic arm.

The complexity and dynamics of the Multi-Agent Systems make it difficult for the system to obtain the gold standard from a human expert. Reinforcement Learning (RL), however, can be introduced to learn the policy by trial and error interaction in order to optimize their task performance by defining reward and penalty mathematically based on the interaction between an agent with the environment the collision-free path [74].

In 2015, Mnih et al. [75] first introduced deep Q-Networks (DQNs), which is a synthesized model of deep convolutional neural networks with the Q-learning algorithm. This model acts as a non-linear state abstraction tool, which deals with action decisions as well as evaluations. This model outperforms the previous approaches that uses a neural network as a function approximation for the Q-function. The experience replay is introduced in DQNs to maintain a buffer to store and update previous experiences consisting of the current state s_t, the chosen action a_t, the reward r_t and the next state s_{t+1}. The mini-batch is applied to update the samples of experience from this buffer. The back-propagation algorithm is applied based on the sampled experiences. The DQNs employed experience replay and target network, which contributes to the convincing performance of stability during the training of

the Q action-value function approximation with deep neural networks. The rise of deep RL has enhanced the capability of agents to perform more complex and dynamic tasks in high dimensional continuous state and action spaces.

MARL has been widely explored to answer the research problem to control the multi-robot teams in both cooperative/competitive game and motion planning. The MARL enables the robot team involved in the tasks to learn and optimise the policy by rewards or payoffs obtained through interacting with their environments during the mapping from their states for their actions. As a rapidly expanding field, a wide variety of approaches have been proposed in the MARL to exploit its benefits and address its challenges during recent decades. These approaches integrate developments in the areas of single-agent RL, game theory, and deep learning.

In recent decades, MARL has been rapidly developed and inspired various approaches to accomplish tasks in the field of multi-robot systems based on the development of single-agent reinforcement learning and game theory. In terms of the difference in reward functions, three kinds of tasks are under consideration in MARL approach, including:

- ***Fully cooperative tasks***: the reward functions are the same among the agents since there is no conflict between their goals;
- ***Fully competitive tasks***: the reward function would be opposite in fully competitive tasks due to completely opposite goals;
- ***Cooperative and competitive tasks***: the reward function doesn't have specific constraint.

Incomplete information, large learning space, and dynamic environments are the three main barriers to learning in multi-robot systems. There is a particular difficulty in scaling the established MARL, i.e. Minimax-Q learning, Nash-Q learning, into MRS, which consists of large and continuous state and action spaces. This will affect the learning performance, including convergence, efficiency, and stability in the approximation and generalisation.

Reinforcement learning with imitation learning

Generally, it is also a good practice to train the policy by imitation learning and improve the performance and convergence rate by reinforcement learning.

Chen et al. [76] proposed a decentralised multi-agent collision avoidance algorithm based on deep reinforcement learning. A value network was developed to encode the estimated time to the goal given an agent's joint configuration (positions and velocities) with its neighbours. After training this value network, the agent obtains the policy to efficiently compute the

velocity vector to avoid collision and also to predict the uncertain motion of the other agents from experience to avoid deadlock. Their result shows that their network can improve 26% in the quality of the path (i.e., time to reach the goal) than ORCA [1].

Sartoretti et al. [15] developed a hybrid algorithm, called PRIMAL, based on reinforcement learning (RL) and imitating from a centralised expert (IL) algorithm. In PRIMAL, an individual agent is learned from a common single-agent policy and trained to consider the consequences of actions on others to achieve optimal performance for the team, and the final learned policy can be copied onto any number of agents. Agents controlled by PRIMAL perform well in various team sizes in low obstacle densities, but overall, the method does not generalise to a large number of agents and higher obstacle densities. Extended on this work, Damani et al [77] proposed a distributed reinforcement learning framework, PRIMAL2, for lifelong multi-agent pathfinding (LMAPF). In this algorithm, agents learn fully decentralized policies to reactively plan paths online in a partially observable world to coordinate considerable agents without any compromise on reactivity and scalability

Multiagent reinforcement learning with communication

Recently, there has been a rising trend to explore the impact of communication (message passing or information sharing) between agents in MARL.

Foerster et al. [78] first developed a network to train agents to encode meaning in discrete communicative actions to solve the 'Switch Riddle' problem. However, their method has limitations when there is a fixed number of agents, and when there is a serial and sequential communication between agents. Sukhbaatar et al. [79] proposed a simple neural model, called CommNet for continuous communication for fully cooperative tasks. This model can be applied in various tasks to train the policy of multi-agents and communication between them. Their results show that the agents can learn to communicate with each other by this model, and obtained improved performance against non-communicate agents.

The multiplayer game is one of the most popular test-beds for the multi-agent algorithm. Peng et al. [80] introduced Multiagent Bidirectionally-Coordinated Network (BiCNet) with a vectorized extension of the actor-critic formulation. Agents can communicate with each other via BiCNet, while the network is trained by a multi-agent actor-critic framework. Various experiments have been carried out on StarCraft to validate the robustness, scalability, and generalization of the proposed network. The results show that the network can handle different types of combats with an arbitrary number of agents and outperform multiple baselines without any supervision, including human demonstration and labeled data. Jaderberg et al. [81] devised a two-tier optimization process, where a number of the independent agents are trained by reinforcement learning concurrently from thousands of parallel matches based

on arbitrary setups, including various maps, number of players, and choice of team. Each agent can be trained by its own internal reward function and rich representation of the world. The proposed framework was validated by tournament-style competition, where the trained policy presented a human-level performance in a three-dimensional multi-player first-person video game, called Capture the Flag.

Wang et al [82] proposed a reinforcement learning-based approach called FCMNet that allows agents to simultaneously learn an effective multi-hop communications protocol and a common, decentralized policy that enables team-level decision-making. The proposed method utilizes the hidden states of multiple directional recurrent neural networks as communication messages among agents. Using a simple multi-hop topology, each agent can receive information sequentially encoded by every other agent at each time step, leading to improved global cooperation. The paper demonstrates the state-of-the-art performance of FCMNet on a challenging set of StarCraft II micromanagement tasks with shared rewards and a collaborative multi-agent path-finding task with individual rewards.

Recent research in machine learning has been attracted to GNNs, which utilize the great expressive power of graphs to handle data with rich relational information among elements [83]. Graph neural networks (GNNs) are connectionist models that capture the dependence of graphs via message passing between the nodes of the graph. Different from Convolutional Neural Networks (CNNs), GNNs retain a state that can represent information from its neighborhood with arbitrary depth. GNNs have applied been applied with considerable success to various areas including social networks [84], the internet of things [85], and natural science (protein-protein interaction network [86]). Therefore, there are some recent attempts to apply GNNs in processing data from dynamic environments or communication for Multi-Agent or multi-robot systems.

Malysheva et al. [87] proposed a framework called MAGnet and validate it in a cooperative and competitive game called Pommerman game. The proposed framework consists of the graph generation stage and decision-making stage, where the first graph-neural network is introduced to process the relationship between agents and between agents and environments represented by a relevant graph. Then state representation is passed into the decision-making stage, which is trained by reinforcement learning to output action for each agent. Their results show that it can obtain a better benchmark than other algorithms, including DQN, MADDPG and MCTS. Besides, Tolstaya et al. [18] proposed a network where the local controllers are trained to mimic the policy of a centralized controller using global information at the training stage and achieve a comparable performance of it with only local information and local communication at interference stage. Aggregation graph neural networks (AGNN) [88] is extended to support the time-varying signal and network of the multi-robot teams, which

makes the local controller able to exploit information from distant teammates based on only local information. Their methods are also validated by training a decentralized flocking controller, where the controller can achieve reasonable performance with only local information via AGNN.

In this chapter, we gave a general classification of analytical approaches to multi-robot motion planning problems, where their strengths and limitations of them have been addressed. Then we presented some state-of-the-art learning-based methods in single-agent and multi-agent motion planning, which can potentially offload the online computation (high dimensional joint space) to an offline learning procedure effectively. The work being done in the area of multi-robot path planning using learning-based methods are inspiring and will attract researchers to further explore this domain. In the next chapter, we will define a specific formulation of the problem and our proposed method to be carried out.

Chapter 3

Decentralized Multi-Robot Motion Planning

3.1 Introduction

Efficient and collision-free navigation in multi-robot systems is fundamental to advancing mobility. The problem, generally referred to as Multi-Robot Path Planning (MRPP) or Multi-Agent Path Finding (MAPF), aims at generating collision-free paths leading robots from their origins to designated destinations.

Our series of works focuses on multi-robot path planning for scenarios where the robots are restricted in observation and communication range, and possess no global reference frame for localization. This naturally arises when considering physical robots equipped with hardware constraints that limit their perception and communication capabilities [89]. These scenarios impose a decentralized structure, where at any given point in time, robots have only partial information of the system state.

In Sec 3.4, we propose a combined architecture, where we train a convolutional neural network (CNN) [90] that extracts adequate features from local observations, and a graph neural network (GNN) to communicate these features among robots [91] with the ultimate goal of learning a decentralized sequential action policy that yields efficient path plans for all robots. The GNN implementation seamlessly adapts to the partial information structure of the problem, since it is computed in a decentralized manner. We train this architecture to imitate an optimal coupled planner with global information that is available offline at training time. Further, we develop a dataset aggregation method that leverages an online expert (CBS algorithm) to resolve hard cases, thus expediting the learning process. The resulting trained model is used online in an efficient, decentralized manner, involving communication only with nearby robots. Furthermore, We achieve performance that is close to that of optimal planners in terms of success rate and flowtime (sum of path costs), while also being able to generalize to previously unseen cases, such as larger robot teams and environments.

As the system scales, decentralized approaches become increasingly popular, where each robot estimates or communicates others' future trajectories via broadcasting or distance-based communication. Unfortunately, if the communication happens concurrently and equivalently among many neighboring robots, it is likely to cause redundant communication, burden the computational capacity and adversely affect overall team performance. Besides, robust and continuous communication cannot yet be guaranteed due to limited bandwidth, large data volumes, and interference from the surroundings. Additionally, under a fully decentralized framework without any priority of planning, it is very hard to ensure the convergence of the negotiation process [12]. These limitations ultimately affect the optimality of solutions found and the overall resilience of the team to disruptions. Hence, new trends of research focus on ***communication-aware path planning approaches*** by explicitly considering communica-

tion efficiency during path generation and path optimization [63], addressing to whom the information is communicated, and at what time [62].

To overcome this, we propose to use a Message Aware Graph Attention neTwork (MAGAT) to extend our previous decentralized framework [19] in Sec 3.5.

- We combine a Graph Neural Network (GNN) with a key-query-like attention mechanism to improve the effectiveness of inter-robot communication. We demonstrate the suitability of applying our model on *dynamic communication graphs* by proving its permutation equivariance and time invariance property.
- We investigate the impact of reduced communication bandwidth by reducing the size of the shared features, and then deploy a skip-connection to preserve self-information and maintain model performance.
- We demonstrate the generalizability of our model by training the model on small problem instances and testing it on increasing robot density, varying map size, and much larger problem instances (up to $\times 100$ the number of robots. Our proposed model is shown to be more efficient in learning general knowledge of path planning as it achieves better generalization performance than the baseline systems under various scenarios. Note that the baseline includes the prior algorithms (HCA and Replanning) and baseline neural network model.

3.2 Related Work

Multi-robot path planning. Classical approaches to multi-robot path planning can generally be described as either centralized or decentralized. *Centralized* approaches are facilitated by a planning unit that monitors all robots' positions and desired destinations, and returns a coordinated plan of trajectories (or way-points) for all the robots in the system. These plans are communicated to the respective robots, which use them for real-time on-board control of their navigation behavior. Coupled centralized approaches, which consider the joint configuration space of all involved robots, have the advantage of producing optimal and complete plans, yet tend to be computationally very expensive. Indeed, solving for optimality is NP-hard [92], and although significant progress has been made towards alleviating the computational load [93, 94], these approaches still scale poorly in environments with a high number of potential path conflicts.

Decentralized approaches provide an attractive alternative to centralized approaches, firstly, because they reduce the computational overhead, and secondly, because they relax the

dependence on centralized units. This body of work considers the generation of collision-free paths for individual robots that cooperate only with immediate neighbors [95, 13], or with no other robots at all [1]. In the latter case, coordination is reduced to the problem of reciprocally avoiding other robots (and obstacles), and can generally be solved without the use of communication. Yet, by taking purely local objectives into account, global objectives (such as path efficiency) cannot be explicitly optimized. In the former case, it has been shown that monotonic cost reduction of global objectives can be achieved. This feat, however, relies on strong assumptions (e.g., problem convexity and invariance of communication graph [96, 97]) that can generally not be guaranteed in real robot systems.

Learning-based methods. *Learning-based* methods have proven effective at designing robot control policies for an increasing number of tasks [98, 99]. The application of learning-based methods to multi-robot motion planning has attracted particular attention due to their capability of handling high-dimensional joint state-space representations, by offloading the online computational burden to an offline learning procedure. The work in [100] proposes a decentralized multi-agent collision avoidance algorithm based on deep reinforcement learning. Their results show that significant improvement in the quality of the path (i.e., time to reach the goal) can be achieved with respect to current benchmark algorithms (e.g., ORCA [1]). Also in recent work, Sartoretti et al. [15] propose a hybrid learning-based method called PRIMAL for multi-agent path-finding that uses both imitation learning (based on an expert algorithm) and multi-agent reinforcement learning. It is note-worthy that none of the aforementioned learning-based approaches consider inter-robot communication, and thus, do not exploit the scalability benefits of fully decentralized approaches. Learning what, how, and when to communicate is key to this aim.

Of particular interest to us is the capability of learning-based methods to handle high-dimensional joint state-space representations, useful when planning for large-scale collective robotic systems, by offloading the online computational burden to an offline learning procedure [18, 101, 100]. The fact that each robot must be able to accumulate information from other robots in its neighborhood is key to this learning procedure. From the point of view of an individual robot, its local decision-making system is incomplete, since other agents' unobservable states affect future values. *The manner in which information is shared is crucial to the system's performance, yet is not well addressed by current machine learning approaches.* Graph Neural Networks (GNNs) promise to overcome this deficiency [102, 91]. They capture the *relational* aspect of robot communication and coordination by modeling the collective robot system as a graph: each robot is a node, and edges represent communication links [103]. Although GNNs have been applied to a number of problem domains, including molecular biology [104], quantum chemistry [105], and simulation engines [106],

they have only very recently been considered within the multi-robot domain, for applications of flocking and formation control [103, 18, 101].

Attention mechanisms. GNNs have attracted increasing attention in various fields, including the multi-robot field (flocking and formation control [18], and multi-robot path planning [19]). However, how to effectively process large-scale graphs with noisy and redundant information is still under active investigation. A potential approach is to introduce attention mechanisms to actively measure the relative importance of node features. Hence, the network can be trained to focus on task-relevant parts of the graph [107]. Learning attention over static graphs has been proved to be efficient. Besides, Liu et al. [108] developed a learning-based communication model that constructed the communication group on a static graph to address what to transmit and which agent to communicate for collaborative perception. However, its permutation equivariance, time invariance and its practical effectiveness in dynamic multi-agent communication graphs have not yet been verified.

3.3 Problem formulation

3.3.1 Problem.

We set up a 2D grid world $\mathscr{W}$, which contains a set of static obstacles $\mathscr{C} \subset \mathscr{W}$. A static obstacle is a two-dimensional polygon. Let $\mathscr{V} = \{v_1, \ldots, v_N\}$ be the set of N robots with independent pairs of the start and goal positions.

We formulate the multi-agent path planning problem as a sequential decision-making problem that each robot solves at every time step t, with the objective of reaching its destination. We assume that communication between robots and decision-making were achieved instantly without delay. In this work, we initially approach the task in a discrete domain, aligning with the subsequent definition of time presented in the follow-up content. More formally, the goal of this work is to learn a mapping $\mathscr{F}$ that takes the maps $\{\mathbf{Z}_t^i\}_{v_i \in \mathscr{V}}$ and the communication network $\mathscr{G}_t$ at time t and determines an appropriate action $\mathbf{u}_t$.

We want the action $\mathbf{u}_t = \mathscr{F}(\{\mathbf{Z}_t^i\}, \mathscr{G}_t)$ to be such that it contributes to the global objective of moving the robots towards their destinations in the shortest possible time while avoiding collisions with other robots and with obstacles that might be present.

The objective is to train the network to perform as well as a coupled centralized expert, while restricting robots to partial observations. The mapping $\mathscr{F}$ has to be restricted to involve communication only among nearby robots, as dictated by the network $\mathscr{G}_t$ at each time instant t.

Finally, note that for scalability, the mapping $\mathscr{F}$ cannot depend on time t, allowing the system to process sequences of arbitrary duration.

3.3.2 Assumptions.

We have made assumptions as follows:

- There is no global positioning of the robots. The map perceived by robot i at time t is denoted by $\mathbf{M}_t^i \in \mathbb{R}^{W_{\text{FOV}} \times H_{\text{FOV}}}$, where W_{FOV} and H_{FOV} are the width and height (in 2D plane) of the rectangular Field of View (FOV).
- When the goal of the robot is outside the FOV, this information is clipped to the FOV boundary in a local reference frame (see Fig. 3.1), resulting in only the awareness of the direction of its goal. This design is inspired by the PRIMAL [15].
- Compared with the time for the robot's movement, the communication between the neighboring robots happens instantly without delay, and is not blocked by any static obstacles $\mathscr{C}$.
- We further limit the communication such that robots can only send out their features at a specific bandwidth, but **cannot** access the ownership of the feature received.

3.3.3 Communications.

The definition of communication is the key difference between our previous work [19] (as summarized in Sec. 3.3.3 and detailed in Sec. 3.4) and our Message-aware Graph Attention Networks (MAGAT) [39] (as summarized in Sec. 3.3.4 and detailed in Sec. 3.5.1 and Sec. 3.5.2), which demonstrated the greater generalized ability.

Vanilla graph neural network for multi-robot path planning

Let $\mathscr{V} = \{v_1, \ldots, v_N\}$ be the set of N robots. At the time t, each robot perceives its surroundings within a given field of vision; although the robot knows where its own target destination is located, this information is clipped to the field of vision in a local reference frame (see Fig. 3.1). Furthermore, we assume no global positioning of the robots. This map perceived by robot i is denoted by $\mathbf{Z}_t^i \in \mathbb{R}^{W_{\text{FOV}} \times H_{\text{FOV}}}$ where W_{FOV} and H_{FOV} are the width and height, respectively, and are determined by the field of vision radius r_{FOV}.

Robots can communicate with each other as determined by the communication network. We can describe this network at time t by means of a graph $\mathscr{G}_t = (\mathscr{V}, \mathscr{E}_t, \mathscr{W}_t)$ where $\mathscr{V}$ is the

set of robots, $\mathscr{E}_t \subseteq \mathscr{V} \times \mathscr{V}$ is the set of edges and $\mathscr{W}_t : \mathscr{E}_t \to \mathbb{R}$ is a function that assigns weights to the edges. Robots v_i and v_j can communicate with each other at time t if $(v_i, v_j) \in \mathscr{E}_t$. The corresponding edge weight $\mathscr{W}_t(v_i, v_j) = w_t^{ij}$ can represent the strength of the communication (or be equal to 1 if we are only modeling whether there is a link or not). For instance, two robots v_i and v_j, with positions $\mathbf{p}_i, \mathbf{p}_j \in \mathbb{R}^2$ respectively, can communicate with each other if $\|\mathbf{p}_i - \mathbf{p}_j\| \leq r_{\text{COMM}}$ for a given communication radius $r_{\text{COMM}} > 0$. This allows us to define an adjacency matrix $\mathbf{S}_t \in \mathbb{R}^{N \times N}$ representing the communication graph, where $[\mathbf{S}_t]_{ij} = s_t^{ij} = 0$ if $(v_j, v_i) \notin \mathscr{E}_t$.

Message graph neural network for multi-robot path planning

Each robot can communicate, or share information with only adjacent robots. We formalize this communication with a dynamic distance-based communication network. We can describe this network at time t by means of a graph $\mathscr{G}_t = (\mathscr{V}, \mathscr{E}_t, \mathscr{W}_t)$ where $\mathscr{V}$ is the set of robots, $\mathscr{E}_t \subseteq \mathscr{V} \times \mathscr{V}$ is the set of edges and $\mathscr{W}_t : \mathscr{E}_t \to \mathbb{R}$ is a function that assigns weights to the edges. The graph is distance-based because robots v_i and v_j can communicate with each other at time t if and only if $(v_i, v_j) \in \mathscr{E}_t$. In addition to vanilla GNN, we introduce the corresponding edge weight $\mathscr{W}_t(v_i, v_j) = w_t^{ij} = [\mathbf{S}_t]_{ij}[E]_{ij} \in [0, 1]$, where $[\mathbf{S}_t]_{ij}$ represents the graph connectivity and $[E]_{ij}$ (will be introduced later in Eq. 3.9) indicates relative importance (attention) of the information contained in the messages received from the neighboring robots.

3.3.4 Preliminaries

In order to guarantee that the mapping $\mathscr{F}$ is restricted to communications only among nearby robots, we parametrize it by means of a GNN, which is a naturally decentralized solution (Sec. 3.3.4). We then train this GNN to learn appropriate actions that contribute to the global objective by means of supervised learning through an expert algorithm (i.e., imitation learning) (Sec. 3.4.1).

Graph Convolutions

Assume that each robot has access to F observations $\tilde{\mathbf{x}}_t^i \in \mathbb{R}^F$ at time t. Let $\mathbf{X}_t \in \mathbb{R}^{N \times F}$ be the observation matrix where each row collects these F observations at each robot $\tilde{\mathbf{x}}_t^i$, $i = 1, \ldots, N$,

$$\mathbf{X}_t = \begin{bmatrix} (\tilde{\mathbf{x}}_t^1)^{\text{Tr}} \\ \vdots \\ (\tilde{\mathbf{x}}_t^N)^{\text{Tr}} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_t^1 & \cdots & \mathbf{x}_t^F \end{bmatrix}. \tag{3.1}$$

Note that the columns $\mathbf{x}_t^f \in \mathbb{R}^N$ represent the collection of the observation f across all nodes, for $f = 1, \ldots, F$. This vector $\mathbf{x}_t^f$ is a *graph signal* [109], since it assigns a scalar value to each node, $\mathbf{x}_t^f : \mathscr{V} \to \mathbb{R}$ so that $[\mathbf{x}_t^f]_i = x_t^{if} \in \mathbb{R}$.

To formally describe the communication between neighboring agents, we need a concise way of describing the graph $\mathscr{G}_t$ and relating it to the observations $\mathbf{X}_t$. Towards this end, we use the adjacency matrix $\mathbf{S}_t$. We note that other matrix descriptions of the graph, such as the Laplacian matrix or the Markov matrix are possible. We generically call $\mathbf{S}_t$ the *graph shift operator* (GSO) [109].

The operation $\mathbf{S}_t\mathbf{X}_t$ represents a linear combination of neighboring values of the signal due to the sparsity pattern of $\mathbf{S}_t$. More precisely, note that the value at node i for observation f after operation $\mathbf{S}_t\mathbf{X}_t \in \mathbb{R}^{N \times F}$ becomes

$$[\mathbf{S}_t\mathbf{X}_t]_{if} = \sum_{j=1}^{N} [\mathbf{S}_t]_{ij}[\mathbf{X}_t]_{jf} = \sum_{j:v_j \in \mathscr{N}_i} s_t^{ij} x_t^{jf} \tag{3.2}$$

where $\mathscr{N}_i = \{v_j \in \mathscr{V} : (v_j, v_i) \in \mathscr{E}_t\}$ is the set of nodes v_j that are neighbors of v_i. Also, the second equality in (3.2) holds because $s_t^{ij} = 0$ for all $j \notin \mathscr{N}_i$.

The linear operation $\mathbf{S}_t\mathbf{X}_t$ is essentially *shifting* the values of $\mathbf{X}_t$ through the nodes, since the application of $\mathbf{S}_t$ updates the value at each node by a linear combination of values in the neighborhood. With the shifting operation in place, we can define a *graph convolution* [91] as linear combination of shifted versions of the signal

$$\mathscr{A}(\mathbf{X}_t; \mathbf{S}_t) = \sum_{k=0}^{K-1} \mathbf{S}_t^k \mathbf{X}_t \mathbf{A}_k \tag{3.3}$$

where $\{\mathbf{A}_k\}$ is a set of $F \times G$ matrices representing the filter coefficients combining different observations. Several noteworthy comments are in order with respect to (3.3). First, multiplications to the left of $\mathbf{X}_t$ need to respect the sparsity of the graph since these multiplications imply combinations across different nodes. Multiplications to the right, on the other hand, can be arbitrary, since they imply linear combination of observations within the same node in a weight sharing scheme. Second, $\mathbf{S}_t^k\mathbf{X}_t = \mathbf{S}_t(\mathbf{S}_t^{k-1}\mathbf{X}_t)$ is computed by means of k communication exchanges with 1-hop neighbors, and is actually computing a summary of the information located at the k-hop neighborhood. Therefore, the graph convolution is an entirely local operation in the sense that its implementation is naturally distributed. Third, the graph convolution is actually computing the output of a bank of FG filters where we take as input F observations per node and combine them to output G observations per node, $\mathscr{A}(\mathbf{X}_t; \mathbf{S}_t) \in \mathbb{R}^{N \times G}$. There are FG graph filters involved in (3.3) each one consisting of K

filter taps, i.e., the (f,g) filter can be described by filter taps $\mathbf{a}^{fg} = [a_0^{fg}, \ldots, a_{K-1}^{fg}] \in \mathbb{R}^K$ and these filter taps are collected in the matrix $\mathbf{A}_k$ as $[\mathbf{A}_k]_{fg} = a_k^{fg}$.

Graph Neural Networks

A convolutional GNN [91] consists of a cascade of L layers, each of which applies a graph convolution (3.3) followed by a pointwise nonlinearity $\sigma : \mathbb{R} \to \mathbb{R}$ (also known as activation function)

$$\mathbf{X}_\ell = \sigma\big[\mathscr{A}_\ell(\mathbf{X}_{\ell-1}; \mathbf{S})\big] \quad \text{for} \quad \ell = 1, \ldots, L \tag{3.4}$$

where, in a slight abuse of notation, σ is applied to each element of the matrix $\mathscr{A}_\ell(\mathbf{X}_{\ell-1}; \mathbf{S})$. The input to each layer is a graph signal consisting of $F_{\ell-1}$ observations and the output has F_ℓ observations so that $\mathbf{X}_\ell \in \mathbb{R}^{N \times F_\ell}$. The input to the first layer is $\mathbf{X}_0 = \mathbf{X}_t$ so that $F_0 = F$ and the output of the last layer corresponds to the action to be taken at time t, $\mathbf{X}_L = \mathbf{U}_t$ which could be described by a vector of dimension $F_L = G$. The GSO $\mathbf{S}$ to be used in (3.4) is the one corresponding to the communication network at time t, $\mathbf{S} = \mathbf{S}_t$. At each layer ℓ we have a bank of $F_\ell F_{\ell-1}$ filters $\mathscr{A}_\ell$ described by a set of $K_\ell F_\ell F_{\ell-1}$ total filter taps $\{\mathbf{A}_{\ell k}\}_{k=0}^{K-1}$.

We note that, in the present framework, we are running one GNN (3.4) per time instant t, where each time step is determined by the moment the action is taken and the communication network changes. This implies that we need to carry out $\sum_{\ell=1}^{L}(K_\ell - 1)$ total communications before deciding on an action. Therefore, it is important to keep the GNN shallow (small L) and the filters short (small K_ℓ).

In summary, we propose to parametrize the mapping $\mathscr{F}$ between maps $\mathbf{Z}_t$ and actions $\mathbf{U}_t$ by using a GNN (3.4) acting on observations $\mathbf{X}_t = \mathrm{CNN}(\mathbf{Z}_t)$ obtained by applying a CNN to the input maps. We note that, by choosing this parametrization we are obtaining a mapping that is naturally distributed and that is adequately exploiting the network structure of the data.

3.4 Graph Neural Networks for Decentralized Multi-Robot Path Planning

3.4.1 Architecture

The following sections describe the architecture, of which all components are illustrated in Fig. 3.1.

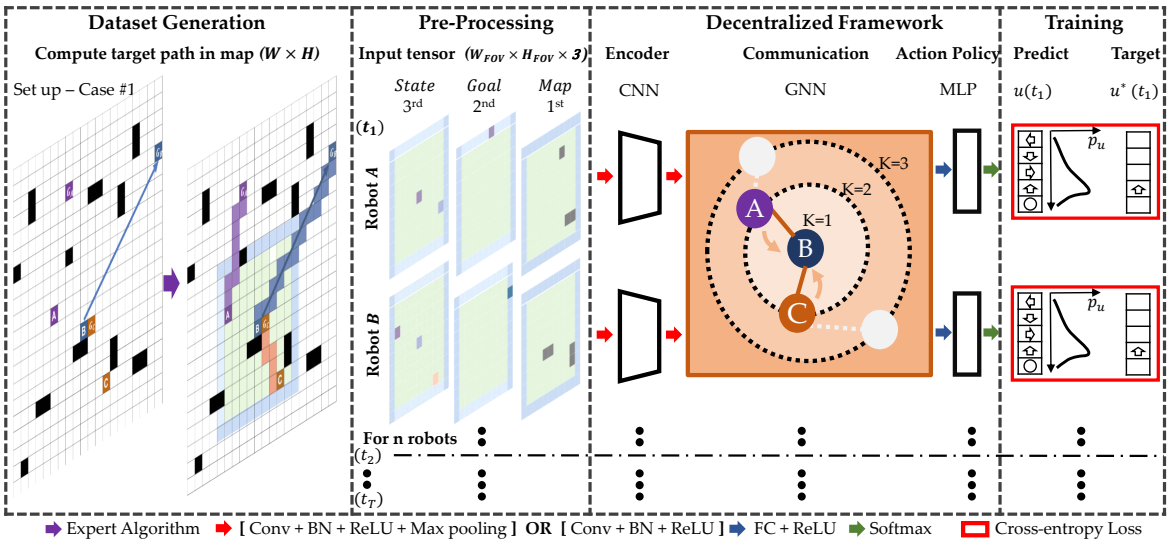


Fig. 3.1 Illustration of the proposed framework. *(i)* The input tensor is based on a binary map representation (1st channel: partial observation of the environment; 2nd channel: the position of goal ($\mathbf{p}^i_{goal}$), or its projection onto the boundary of the field-of-view; 3rd channel: self (agent) at center, with other agents within its field-of-view). *(ii)* The decentralized framework consists of a CNN to extract observations from the input tensor, a GNN to exchange information between the neighboring agents, and an MLP to predict the actions. *(iii)* Training is performed through cross-entropy loss over a discrete action space.

Processing Observations

In an environment ($W \times H$) with static obstacles, each robot has a local field-of-view (FOV), the radius of which is defined by r_{FOV}, beyond which it cannot 'see' anything. Inspired by the observation space in PRIMAL [15], we set the robot itself located at the center of this local observation without prior knowledge of its global position. The data available at robot i is a map $\mathbf{Z}^i_t$ of size $W_{\text{FOV}} \times H_{\text{FOV}}$ (Fig. 3.1 illustrates how we implement such partial observations). The input map $\mathbf{Z}^i_t$ is fed into a CNN that is run internally on each robot. This results in a vector $\tilde{\mathbf{x}}^i_t \in \mathbb{R}^F$ containing F observations (3.1), $\tilde{\mathbf{x}}^i_t = \text{CNN}(\mathbf{Z}^i_t)$. These observations can then be communicated to nearby robots. The intuition behind using a CNN is to process the input map $\mathbf{Z}^i_t$ into a higher-level feature tensor $\tilde{\mathbf{x}}^i_t$ describing the observation, goal and states of other robots. This feature tensor is then transmitted via the communication network, as described in the following section,Sec. 3.4.1.

Communication

Each individual robot communicates its compressed observation vector $\tilde{\mathbf{x}}^i_t$ with neighboring robots within its communication radiGraph Neural Networks for Decentralized Multi-

Robotus r_{COMM} over the multi-hop communication network, whereby the number of executed hops is limited by K. As described in Sec. 3.3.4, we apply our GNN to aggregate and fuse the states $(\tilde{\mathbf{x}}_t^j)$ within this K-hop neighborhood of robots $j \in \mathcal{N}_i$, for each robot i. The output of the communication GNN is a hyper-representation of the fused information of the robot itself and its K-hop neighbors, which is passed to the action policy, as described in Sec. 3.4.1. We note that each robot carries a local copy of the GNN, hence resulting in a localized decision-making policy.

Action Policy

We formulate the path-finding problem as a sequential classification problem, whereby an optimal action is chosen at each time step. We adopt a local multi-layer perceptron (MLP) to train our action policy network. More specifically, each node applies a MLP to the aggregated features resulting from the communication GNN. This MLP is the same across all nodes, resembling a weight-sharing scheme. The action $\tilde{\mathbf{u}}_t^i$ taken by robot i is given by a stochastic action policy based on the probability distribution over motion primitives, which in our case consists of five discrete options (up, left, down, right, idle), and are represented by one-hot vectors. The final path is represented by the series of sequential actions.

Network Architecture

We construct our CNN architecture by using `Conv2d-BatchNorm2d-ReLU-MaxPool2d` and `Conv2d-BatchNorm2d-ReLU` blocks sequentially three times. All kernels are of size 3 with a stride of 1 and zero-padding. In the GNN architecture, we deploy a single layer GNN (as described in Sec. 3.3.4) and set 128 as the number of input observations F and output observations G. Note that we can tune the filter taps K for non-communication ($K = 1$) and multi-hop communication ($K > 1$). In the action policy, we use a linear soft-max layer to decode the output observations G from the GNN with 128 features into the five motion primitives.

Learning from Expert Data

To train our models, we propose a supervised learning approach based on expert data (i.e., imitation learning). We assume that, at training time, we have access to an optimal trajectory of actions $\{\mathbf{U}_t^*\}$ for all the robots, and the corresponding maps obtained for this trajectory $\{\mathbf{Z}_t^i\}$, collected in a training set $\mathscr{T} = \{(\{\mathbf{U}_t\}, \{\mathbf{Z}_t^i\})\}$. Then, we train the mapping $\mathscr{F}$ so that the output is as close as possible to the corresponding optimal action $\mathbf{U}^*$ using a cross-entropy loss $\mathscr{L}(\cdot,\cdot)$. If the mapping $\mathscr{F}$ is parametrized in terms of a GNN (3.4) then this optimization

problem becomes

$$\min_{\text{CNN},\{\mathbf{A}_{\ell k}\},\text{MLP}} \sum_{(\{\mathbf{U}_t\},\{\mathbf{Z}_t^i\})\in\mathscr{T}} \sum_{t\in[0,T_{\text{MP}^*}]} \mathscr{L}(\mathbf{U}_t^*, \mathscr{F}(\{\mathbf{Z}_t^i\},\mathscr{G}_t)). \tag{3.5}$$

where T_{MP^*} is the makespan (the time period from the first robot moves to the last robot arrives its goal) of the expert solution.

We are optimizing over the filters in the CNN required to process the map as well as the set of matrices $\{\mathbf{A}_{\ell k}\}$ that contains the $\sum_{\ell=1}^{L} K_\ell F_{\ell-1} F_\ell$ learnable parameters of the communication GNN. Note that the number of parameters is independent of the size of the network N.

Imitation learning rests on the availability of an optimal solution (further elaborated in the section 'Expert Data Generation', below). While this solution might be computationally expensive, or even intractable for large networks, we only need it at training time. Once trained, the GNN models can be deployed in different communication topologies [110], including those with a larger number of robots as is evidenced in the numerical experiments of Sec. 3.4.2. Given the decentralized nature of the parametrizations, the trained models are efficient in the sense that their computation is distributed among the agents, demanding only communication exchanges with one-hop neighbors.

Expert Data Generation

As described in our problem statement in Sec. 3.3, the robots operate in a grid world of size $W \times H$ with static obstacles randomly placed throughout the map. For each grid world, we generate *cases* randomly, i.e., problem instances, which consist of pairs of start and goal positions for all robots (we also refer to this as a *configuration*). We filter duplicates, invalid cases, or unsolvable cases, and store the remaining cases in a setup pool, which is randomly shuffled at training time. For each case, we generate the optimal solution. Towards this end, we run an expert algorithm: Conflict-Based Search (CBS) [93] (which is a similar approach as taken in [15] and explanation can be found in Sec. 5.3.2). This expert algorithm computes our 'ground-truth paths' (the sequence of actions for individual robots), within a $300\,s$ timeout, for a given initial configuration. Our data set comprises 30,000 cases for any given grid world and number of agents. This data is divided into a training set (70%), a validation set (15%), and a testing set (15%).

Algorithm 1: Generation of sequential actions.

Input: Input tensor, $\mathbf{x}_0^i, i \in [0,N]$, N is the number of robots; timeout $T_{max} = 3T_{\text{MP}^*}$ as explained in Sec. 3.4.2; Policy π

Output: Predicted paths ($\hat{\chi}_i$) for each robot (i), consisting of sequential predicted actions $\hat{u}_t^i$, for all $t \in [0, T_{\text{MP}}]$ from initial position $\mathbf{p}_0^i$

for *t in* $[0, T_{max}]$ **do**
  while *not all robots at their goals* **do**
    for *robot* $i \in \{1, \ldots, N\}$ **do**
      obtain input tensor $\mathbf{x}_t^i$ and adjacency matrix $\mathbf{S}_t$;
      $\hat{u}_t^i \leftarrow \pi(\mathbf{x}_t^i, \mathbf{S}_t)$;
      if *robot i with action* $\hat{u}_t^i$ *collides with obstacle* **then**
        $\hat{u}_t^i \leftarrow$ `idle` (collision shielding);
      end
    end
    if *robot i, with action* $\hat{u}_t^i$*, performs an edge collision with robot j* **then**
      $\hat{u}_t^i \leftarrow$ `idle` (collision shielding);
    else
      record and update position $\mathbf{p}_{t+1}^i$ of robot i by $\hat{u}_t^i$; update input tensor $\mathbf{x}_t^i$ and adjacency matrix $\mathbf{S}_t$.
    end
  end
end
Evaluate $\hat{\chi}$ according to metrics (Sec. 3.4.2).

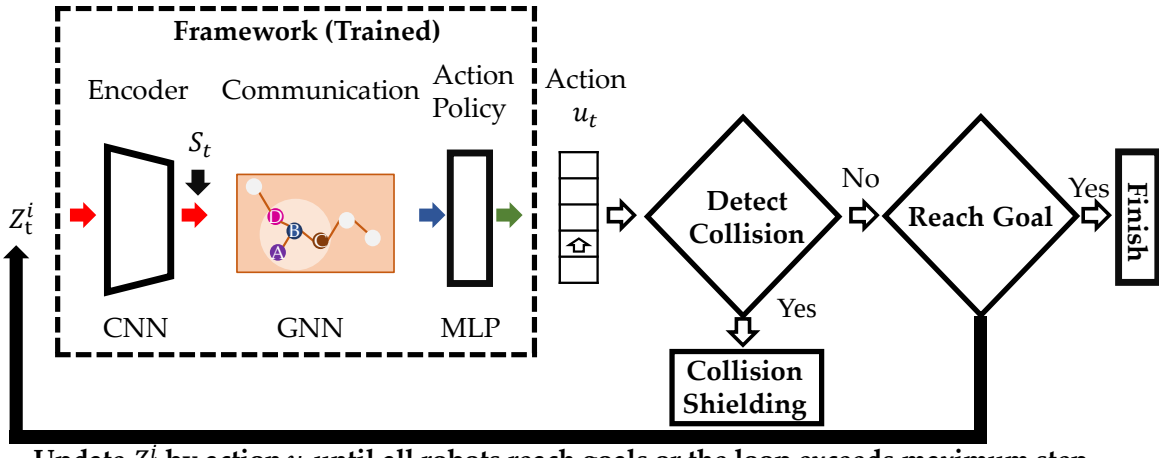


Fig. 3.2 Illustration of the inference stage: for each robot, the input map $\mathbf{Z}_t^i$ is fed to the trained framework to predict the action; collisions are detected and prevented by collision shielding. The input map $\mathbf{Z}_t^i$ is continuously updated until the robot reaches its goal or exceeds the timeout T_{max}.

Policy Execution with Collision Shielding

Inspired by ideas from the community [15, 111], we execute the action policy with a protective mechanism that we name *collision shielding* at the inference stage. Since it is not guaranteed that robots learn collision-free paths, we require this additional mechanism to guarantee that no collisions take place. Collision shielding is implemented as follows: *(i)* if the inferred action would result in a collision with another robot or obstacle, then that action is replaced by an idle action; *(ii)* if the inferred actions of two robots would result in an edge collision (having them swap positions), then those actions are replaced by idle actions. It is entirely possible that robots remain stuck in an idle state until the timeout is reached. When this happens, we count it as a failure case. The overall inference process is summarized in Alg. 1 and Fig. 3.2.

Algorithm 2: Training process with dataset aggregation.

Input: Input tensor, $\mathbf{x}_t^i, t \in [0, T_{\text{MP}^*}], i \in [0, N]$, N is the number of robots; and adjacency matrix $\mathbf{S}_t$; target actions $u_t^{*,i}$ generated expert algorithm; cross-entropy loss $\mathscr{L}$; learning rate γ; $(\mathbf{x}_t^i, \mathbf{S}_t, u_t^{*,i}) \in$ offline dataset D_{offline}

Output: Proposed framework $\pi(\cdot : w)$

```
D ← D_offline ;
π(· : w) ← initialize parameters w ;
for epoch ∈ {1, ..., epoch_max} do
    for {s_t^i, S_t, u_t^{*,i}}_{i=1}^N ∈ D do
        for i ∈ {1, ..., N} do
            û_t^i = π(x_t^i, S_t : w) ;
            w ← w − γ · ∇_w L(û_t^i, u_t^{*,i})
        end
    end
    if mod (epoch, C) = 0 then
        for n_OE randomly selected cases from D_offline do
            Deploy π(· : w) based on Alg. 1;
            Upon timeout, deploy expert algorithm to solve failure case D_OE ;
            D ← D ∪ D_OE
        end
    end
end
```

Dataset Aggregation during Training

The use of collision shielding leads to failure cases due to potential deadlocks in the actions taken by the robots, where some of them remain stuck in an idle state. To overcome such deadlocks, we propose a dataset aggregation method that makes use of an *online expert* (OE)

algorithm, during training. More specifically, every C epochs, we select n_{OE} random cases from the training set and identify which ones are stuck in a deadlock situation. Then, we run the expert starting from the deadlock configuration in order to unlock them into moving towards their goal. The resulting successful trajectory is added to the training set and this extended training set is then used in the following epochs. This process is detailed in Alg. 2. We note that no change is made to the validation or test sets. This dataset aggregation method is similar to the approach in DAgger [112], but instead of correcting every failed trajectory, we only correct trajectories from a randomly selected pool of n_{OE} cases, as calls to our expert algorithm are time-consuming. Another key difference is that we need to resort to an explicit measure of failure (i.e., through the use of a timeout), since focusing on any deviations from the optimal path (as in the DAgger approach) may be misleading, because those paths may still lead to very competitive solutions in our problem setting.

3.4.2 Performance Evaluation

To evaluate the performance of our method, we perform two sets of experiments, *(i)* on networks trained and tested on the *same* number of robots, and *(ii)* on networks trained on a given number of robots, and tested on *previously unseen* team sizes (both larger and smaller).

Metrics

We consider two key performance metrics:

Success Rate (α)

A case is considered successful (*complete*) when *all* robots reach their goal prior to the timeout, i.e., when all robots find their paths from $\mathbf{p}_0^i$ to $\mathbf{p}_{goal}^i$ for $i \in [0,N]$. The success rate is hence quantified by the proportion of successful cases over the total number of tested cases n:

$$\alpha = \frac{n_{\text{success}}}{n} \tag{3.6}$$

Flowtime Increase (δ_{FT})

At the end of the system's inference stage (see Fig. 3.2), the sequence of actions result in a final path, for each robot. The sum of the executed path lengths (FT) may be larger than the sum of expert (target) path lengths (FT*). This deterioration is computed as

$$\delta_{\text{FT}} = \frac{\text{FT} - \text{FT}^*}{\text{FT}^*}. \tag{3.7}$$

Note that if a robot does not reach its goal, the length of the predicted path is considered to be the length of the maximum allowed path length ($T_{max} = 3T_{\text{MP}^*}$)(Alg. 1 and Fig. 3.2). Here, T_{MP^*} is the makespan of the solution generated by the expert algorithm. We also note that computing the flowtime increase with respect to an expert algorithm requires that we can actually solve a case using the expert algorithm in tractable time.

Experimental Setup

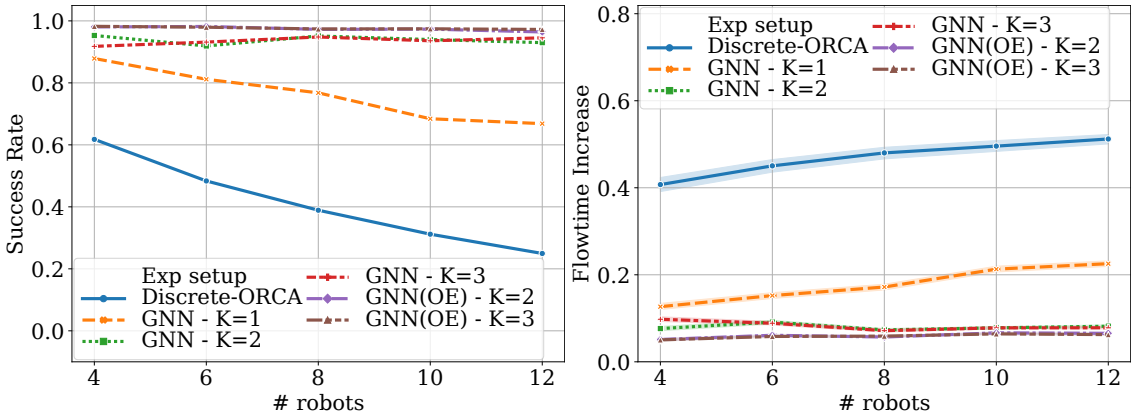
Our simulations were conducted using a 12-core, 3.2Ghz i7-8700 CPU and an Nvidia GTX 1080Ti GPU with 32 and 11GB of memory, respectively. The proposed network was implemented in PyTorch v1.1.0 [113], and was accelerated with Cuda v10.0 APIs. We used the Adam optimizer with momentum 0.9. The learning rate γ scheduled to decay from 10^{-3} to 10^{-6} within 150 epochs, using cosine annealing. We set the batch size to 64, and L2 regularization to 10^{-5}. The online expert on the GNN is deployed every $C = 4$ epochs on $n_{\text{OE}} = 500$ randomly selected cases from the training set.

Results

We instantiate 600 different maps of size 20×20, of which 420 are used for training, 90 for validation, and 90 for testing. We generate 50 cases for each map. The obstacle density is set to 10%, corresponding to the proportion of occupied over free space in the environment. We consider a field of view of radius $r_{\text{FOV}} = 4$ and a communication radius of $r_{\text{COMM}} = 5$. At each time step, each robot runs a forwards pass of its local action policy (i.e., the trained network). At the end of each case (i.e., it is either solved or the timeout is reached), we record the length of each robot's path and the number of robots that reach their goals, to compute performance metrics according to Sec. 3.4.2.

Effect of Communication on Flowtime and Success Rates

Figures 3.3a and 3.3b show results for the success rate and flowtime increase, respectively, as a function of the number of robots. For each panel, we train a model for $N \in [4,6,8,10,12]$, and test it on instances of the same robot team size. In each experiment, we vary the number of communication hops ($K \in [1,2,3]$). Note that for $K = 1$ there is no communication involved. Similar to [1] and [2], we use a discrete version of a velocity-based collision-avoidance method (Discrete-ORCA) as an additional benchmark against which to test our method.



(a) Success rate (α)

(b) Flowtime increase (δ_{FT})

Fig. 3.3 Results for success rate (α) and flowtime increase (δ_{FT}), as a function of the number of robots. For each panel, we vary the number of communication hops ($K \in [1,2,3]$), including results obtained through training with the online expert (OE). We also compare our framework with Discrete-ORCA [1][2].

In both figures, we see a drop in performance for larger teams, but this drop is much more pronounced for the non-communicative GNN ($K = 1$). Our framework generally outperforms the Discrete-ORCA in terms of success rate and flowtime increase.

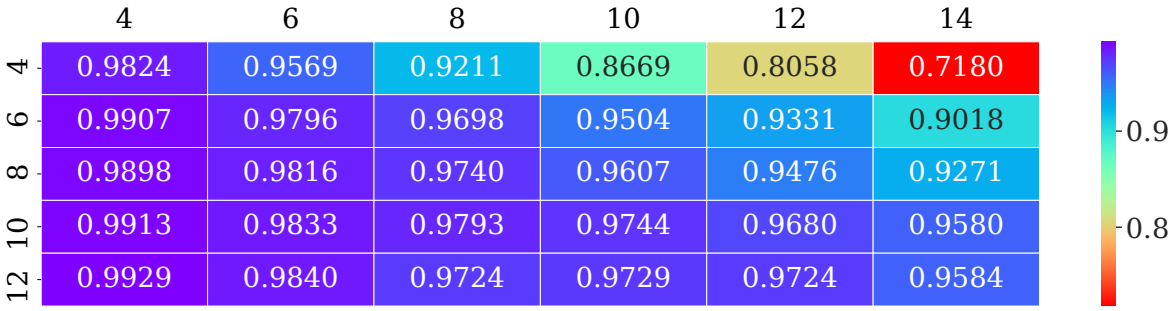
Generalization

Fig. 3.4a and 3.4b summarize the generalization capability of our model for success rate and flowtime increase, respectively. The experiment was carried out by testing networks across previously unseen cases. The tables specify the number of robots *trained on* in the rows, and the number of robots *tested on* in the columns. The results demonstrate strong generalization capabilities.

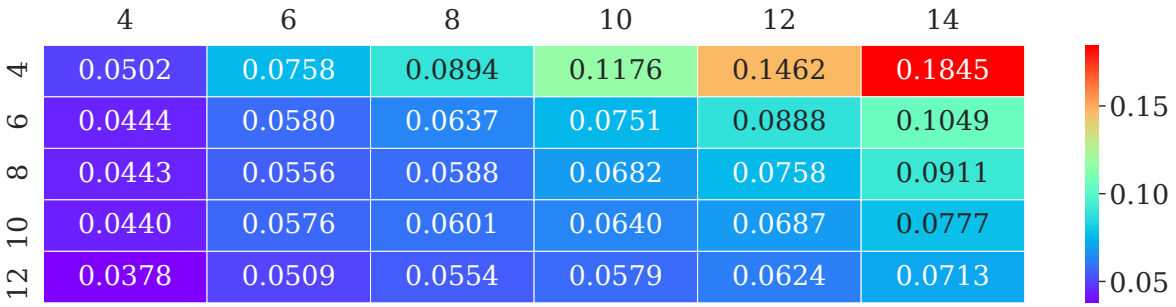
We perform subsequent experiments on larger robot teams to further test the generalization. Results in Fig. 3.5 show that our network, trained on only 10 robots scales to teams of sixfold size. We test the network in different grid map, where the map sizes are scaled to preserve the effective robot density. Notably, the results show *no* degradation of performance.

We train the GNN ($K \in [2,3]$) with and without the online expert (OE) implementation on 10 robots, and test it on 60 robots in 50×50 environments, respectively. The grid maps are scaled to preserve the same effective density $\beta = \frac{n_{\text{robots}}+n_{\text{obs}}}{W \times H}$, where the number of obstacles $n_{obs} = \rho \times W \times H$, ρ is the obstacle density in the map ($W \times H$) and n_{obs} is the number of robots.

Different from our success rate metric, which only considers complete cases (all robots reach their goals), Fig. 3.6 presents the proportion of cases distributed over the number of



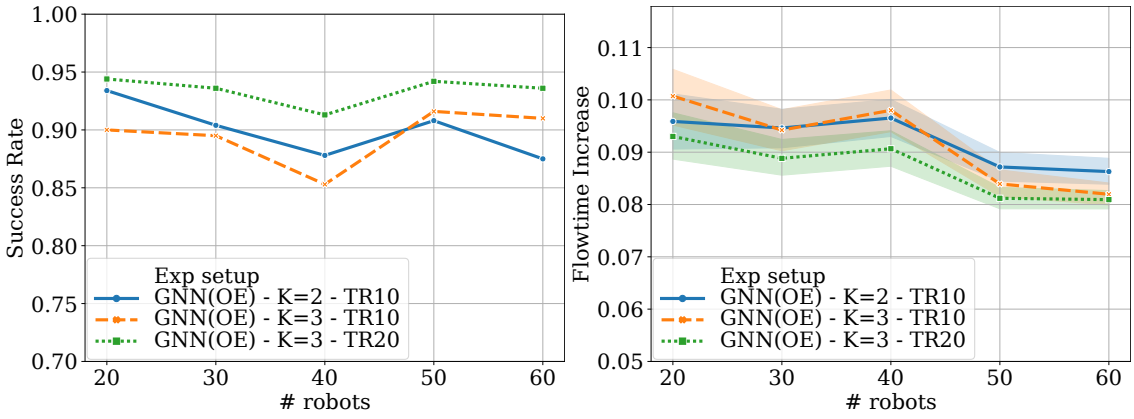
(a) Success rate α for GNN(OE) with K=3



(b) Flowtime increase δ_{FT} for GNN(OE) with K=3

Fig. 3.4 Success rate and flowtime increase. The rows represent the number of robots on which each model was trained, and columns represent the number of robots at test time. The heatmap maps performance to a color range where purple indicates the best performance and red indicates the worst performance.

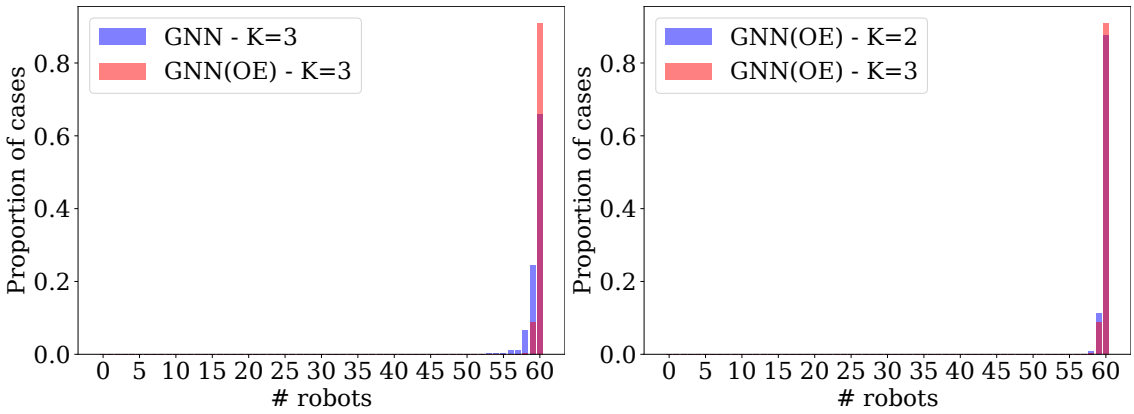
robots reaching their goals. The distributions show that more than 75% of all robots *always* reach their goals across all implementations. In 97% of cases, more than 95% of robots (57 out of 60) reach their goals. For instance, there are 995 out of 1000 cases (99.5%), where at least 54 robots reach their goals with the GNN ($K = 3$) without OE implementation (worst implementation). We see from Fig. 3.6a how the GNN network with OE tends to generalize better than the GNN without OE, since the proportion of robots reaching the goal is larger. In Fig. 3.6b, we see how an increased communication hop count (from $K = 2$ to $K = 3$) contributes to a slightly larger proportion of robots reaching their goals.



(a) Success rate (α)

(b) Flowtime increase (δ_{FT})

Fig. 3.5 Results for success rate and flowtime increase, as a function of the number robots tested on. We vary the GNN implementation ($K \in [2,3]$), trained ('TR10') on a 20×20 map with 10 robots, and GNN implementation ($K = 3$) trained ('TR20') on a 28×28 map with 20 robots. Testing was performed on maps that maintain constant effective robot density.



(a) Effect of online expert

(b) Effect of K

Fig. 3.6 Histogram of proportion of cases distributed over the number of robots reaching their goal; the network with hop count $K \in [2,3]$, is trained on 10 robots and tested on 60 robots, with and without the OE.

Summary

Our results show that the decentralized framework generalizes to different numbers of robots, as seen in Sec. 3.4.2 and Sec. 3.4.2. We note that a single forward pass of our model (enabling a robot in a team of 10 robots to predict its action) takes only about $0.019 \pm 2.15e^{-3}\,s$ on the workstation described in Sec. 3.4.2. In addition to the decentralized nature of our solution, this speed of computation is beneficial in real-world deployments, where each robot runs its own (localized) action policy. In contrast, the expert algorithm [93] is intractable for more than 14 agents in dense environments within the given timeout; this is corroborated by results in [14, 15].

The experiments in Sec. 3.4.2 showed the capability of our decentralized policy to generalize to robot teams across different sizes. Fig. 3.4a and Fig. 3.4b showed that the framework trained in smaller robot teams ($n = 4,\ 6$) tends to perform worse than those trained in larger teams ($n = 8,\ 10,\ 12$), across any unseen instances (larger as well as smaller in size). The intuition for the cause of this phenomenon can be due to two main factors. Firstly, larger robot teams tend to cause more collisions, allowing the policy to learn how to plan more efficient paths more quickly. Secondly, policies trained on very small robot teams (e.g. 4 robots), tend to produce communication topologies that are idiosyncratic, and hence, may generalize more poorly. Results in Fig. 3.5 showed very strong generalization capabilities, with tests scaling to a factor of 6x of the instances trained on, without noticeable performance deterioration.

We also demonstrated that the use of our online expert leads to significant improvements (as seen in Fig. 3.3). Fig. 3.6a shows how the GNN with the online expert was able to increase the success rate of all 60 robots reaching goal given a framework trained on 10 robots, and contribute to a right-shift of the distribution.

3.5 Message-Aware Graph Attention Networks for Large Scale Multi-Robot Path Planning

3.5.1 Message-Aware Graph Attention neTwork

In this work, we initially approach the task in a discrete domain, aligning with the subsequent definition of time presented in the follow-up content. Inspired by the Graph Attention neTworks (GATs) used on static knowledge graphs [107], we incorporate a key-query-like attention mechanism [114] such that the weights on edges between nodes $\mathscr{W}_t(v_i, v_j) \in [0,1]$ are determined by the relative importance of node features, which allows each robot to aggregate message features received from neighbors with a selective focus. Formally, inspired by [115], we define a generic GNN model as follows (affording flexibility to use different weighing mechanism between neighboring nodes):

$$\mathscr{A}(\mathbf{X}_t;\mathbf{S}_t) = \sum_{k=0}^{K-1} (\mathbf{E} \odot \mathbf{S}_t)^k \mathbf{X}_t \mathbf{A}_k \ , \tag{3.8}$$

where $\mathbf{E}$ is an attention matrix of the same dimensions as $\mathbf{S}$ and "$\odot$" refers to an element-wise product. The values of $\mathbf{E}$ are computed as follows:

$$[E]_{ij} = \frac{\exp\left(LeakyReLU(e_{ij})\right)}{\sum_{k\in\mathscr{N}_i} \exp\left(LeakyReLU(e_{ik})\right)} \ , \tag{3.9}$$

where $\mathscr{N}_i$ is the collection of all the neighboring nodes of node i, and e_{ij} is obtained by:

$$e_{ij} = \tilde{\mathbf{x}}_t^i W (\tilde{\mathbf{x}}_t^j)^{\mathrm{Tr}} \ , \tag{3.10}$$

where W is a weight matrix serving as a key-query-like attention [114]. A softmax function (as in Eq. 3.9) is applied to the attention so that the edge weights are constrained within $[0,1]$. Recall that $\tilde{\mathbf{x}}_t$ is the input feature extracted by previous layers.

Similar to GNN, MAGAT generates output features based on a pointwise nonlinearity σ, but potentially the output can be a concatenation of the outputs of P attention heads:

$$\mathbf{X}_\ell = \Big\|_{p=1}^{P} \big(\sigma\big[\mathscr{A}_\ell^p(\mathbf{X}_{\ell-1};\mathbf{S})\big]\big) \quad \text{for} \quad \ell = 1,\ldots,L \ , \tag{3.11}$$

where P is the number of independent heads in the layer and $||$ represents concatenation. Each trainable weight matrix (e.g. W) in each attention head p and each layer l is independent.

We introduce the original GAT [107] as a baseline in Sec. 3.4.2 by directly replacing our core attention mechanism (Eq. 3.10) with the following original GAT attention mechanism

while keeping other parts of our framework unchanged:

$$e_{ij} = ((\tilde{\mathbf{x}}_t^i)^{\mathrm{Tr}}\mathbf{A}_k||(\tilde{\mathbf{x}}_t^j)^{\mathrm{Tr}}\mathbf{A}_k)\mathbf{H}, \tag{3.12}$$

where $\mathbf{H}$ is a $2G_l \times 1$ matrix and $||$ represents concatenation. Note that in the original GAT, the trainable linear-transformation matrix $\mathbf{A}_k$ serves both in the attention weight computation (Eq. 3.12) and the feature aggregation (Eq. 3.8). However, we posit that computing attention weights by Eq. 3.10 on the raw features extracted by the CNN instead of the linear-transformed features, free $\mathbf{A}_k$ from serving two purposes simultaneously, and therefore improves the model performance (as shown in Sec. 3.5.4).

3.5.2 Properties of MAGAT

Given our task formulation, the topology of the communication graph $\mathscr{G}_t$ changes with time. Other robots can enter and leave the communication range of the robot at any time, leading to a frequent change of the graph topology. Therefore, it is necessary to discuss whether our trained MAGAT performs graph convolutions consistently regardless of agent permutation and time shift.

Permutation Equivariance

MAGAT must satisfy permutation equivariance, which ensures that the trained MAGAT is resistant to the change of robot orders and always gives the same convolution results regardless of how we swap the order indices when constructing the dynamic graph.

We first define a permutation π as swapping the indices of robots. The permutation results in a swapped order of features:

$$\pi(\mathbf{X}_t) = \begin{bmatrix} (\tilde{\mathbf{x}}_t^{\pi^{-1}(1)})^{\mathrm{Tr}} \\ \vdots \\ (\tilde{\mathbf{x}}_t^{\pi^{-1}(N)})^{\mathrm{Tr}} \end{bmatrix}. \tag{3.13}$$

A permutation matrix $\mathbf{P}_\pi \in \{0,1\}^{N\times N}$ is thus defined to swap graph features directly:

$$\Big[\mathbf{P}_\pi \mathbf{X}_t\Big]_{ij} = \Big[\mathbf{X}_t\Big]_{\pi^{-1}(i)j}. \tag{3.14}$$

Lemma 1 *Given a permutation π, its corresponding permutation matrix P_π, and the convolution operation $\mathscr{A}_G$ of GNN defined in Eq. 3.3, the following equation can be derived*

from [110]:

$$\mathbf{P}_\pi \mathscr{A}_G(\mathbf{X}_t;\mathbf{S}_t) = \mathscr{A}_G(\mathbf{P}_\pi \mathbf{X}_t;\mathbf{P}_\pi \mathbf{S}_t)\ . \tag{3.15}$$

Proposition 1 (Permutation Equivariance of MAGAT) *For any permutation π, its corresponding permutation matrix P_π and the convolution operation $\mathscr{A}_F$ of MAGAT defined in Eq. 3.8, the following equation holds:*

$$\mathbf{P}_\pi \mathscr{A}_F(\mathbf{X}_t;\mathbf{S}_t) = \mathscr{A}_F(\mathbf{P}_\pi \mathbf{X}_t;\mathbf{P}_\pi \mathbf{S}_t)\ . \tag{3.16}$$

Proof of Proposition 1 Recall that Eq. 3.10 implies that attention e_{ij} is only determined by the node features $\tilde{\mathbf{x}}_t^i$ and $\tilde{\mathbf{x}}_t^j$. Thus, using the permutation operation defined in Eq. 3.14, we can permute the robot indices in the attention matrix as follows:

$$\Big[\mathbf{P}_\pi \mathbf{E}\Big]_{ij} = \text{softmax}(e_{\pi^{-1}(i)j}) = \Big[\mathbf{E}\Big]_{\pi^{-1}(i)j}\ . \tag{3.17}$$

This means that the swap of robots will only permute the attention matrix in a similar way with graph features. Then we can show that in our MAGAT:

$$\begin{aligned}\mathscr{A}_F(\mathbf{P}_\pi \mathbf{X}_t;\mathbf{P}_\pi \mathbf{S}_t) &= \sum_{k=0}^{K-1} (\mathbf{P}_\pi \mathbf{E} \odot \mathbf{P}_\pi \mathbf{S}_t)^k \mathbf{P}_\pi \mathbf{X}_t \mathbf{A}_k \\ &= \sum_{k=0}^{K-1} (\mathbf{P}_\pi (\mathbf{E} \odot \mathbf{S}_t))^k \mathbf{P}_\pi \mathbf{X}_t \mathbf{A}_k \\ &= \mathbf{P}_\pi \sum_{k=0}^{K-1} (\mathbf{E} \odot \mathbf{S}_t)^k \mathbf{X}_t \mathbf{A}_k\ .\end{aligned} \tag{3.18}$$

The last step uses the permutation equivariance property of GNN from *Lemma 1*, taking $\mathbf{E} \odot \mathbf{S}_t$ as a whole to replace the $\mathbf{S}_t$ in GNN.

Time Invariance

MAGAT must satisfy the time invariance criterion, in order to generate consistent output when the same situation appears again in a different step of the simulation.

Proposition 2 (Time Invariance of MAGAT) *Given $t_1 \neq t_2$, $\mathbf{X}_{t_1} = \mathbf{X}_{t_2}$ and $\mathbf{S}_{t_1} = \mathbf{S}_{t_2}$, and the convolution operation $\mathscr{A}_F$ of MAGAT in Eq. 3.8, the following equation holds:*

$$\mathscr{A}_F(\mathbf{X}_{t_1};\mathbf{S}_{t_1}) = \mathscr{A}_F(\mathbf{X}_{t_2};\mathbf{S}_{t_2})\ . \tag{3.19}$$

Proof of Proposition 2 This criterion is satisfied intrinsically by our imitation strategy (Sec. 3.5.3): the decentralized framework is trained to enable the robot to predict consistent

action $\mathbf{U}_t^{*i}$ only with respect to input tensor $\mathbf{Z}_t^i$ and communication network $\mathbf{S}_t$ regardless of the time instant t.

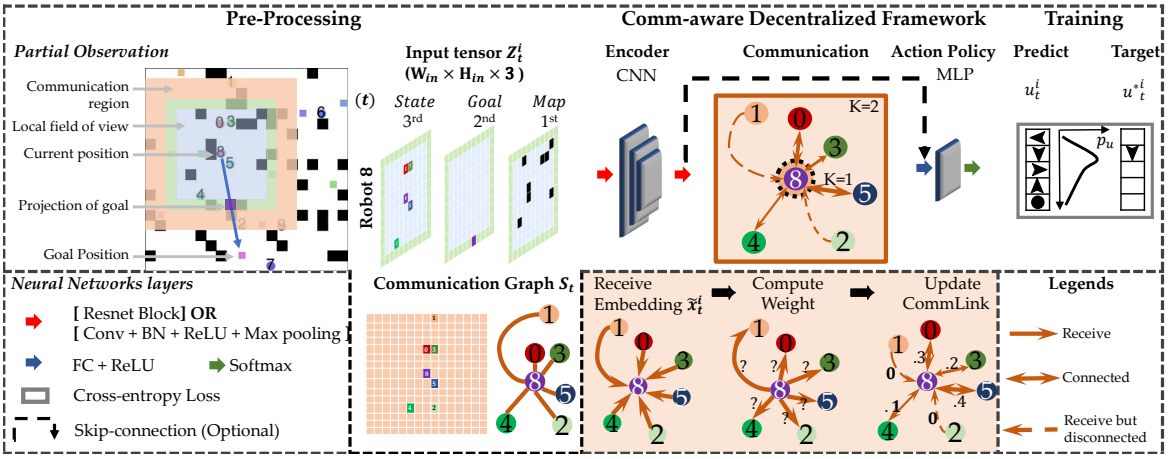


Fig. 3.7 Our proposed decentralized framework. (i) illustrates how we process the partial observations of each robot into input tensor $\mathbf{Z}_t^i$, and how we construct the dynamic communication network. (ii) demonstrates the processing pipeline consisting of a feature extractor, a graph convolution module, and an Multi-layer Perceptron (MLP). The optional skip connection represents the bottleneck structure discussed in Sec. 3.4.1. (iii) visualizes how our model gathers features, computes attention weights by a key-query-like attention mechanism (sandy brown), and selectively aggregates useful features.

3.5.3 Architecture

In this section, we start by introducing the dataset creation (Sec. 3.5.3), and move on to show how we process the observations (Sec. 3.4.1) and the detailed architecture of our proposed model (Sec. 3.4.1). Finally, we present the training process, which is enhanced by an online expert (Sec. 3.5.3).

Dataset Creation

We generate grid worlds of size $W \times H$ and randomly place static obstacles of a given density. For each grid world, we generate *cases* where the start positions and goal positions are not duplicated. We filter duplicates and then call an expert algorithm to compute solutions. Invalid cases (those who do not have a solution, e.g., robots are trapped by static obstacles) are removed at this stage. Towards this end, we run a coupled centralized expert algorithm: Enhanced Conflict-Based Search (ECBS) [1] with optimality bound 1.1 [14]. This expert

[1] https://github.com/whoenig/libMultiRobotPlanning

algorithm computes our 'ground-truth paths' (the sequence of actions for individual robots) for a given initial configuration. Our data set comprises 30,000 cases for any given map size and number of robots. This data is divided into a training set (70%), a validation set (15%), and a testing set (15%). The three sets do not overlap with each other.

Processing Observations

Limited by the local FOV, each robot perceives an observation and processes it to be $\mathbf{Z}_t^i \in \mathbb{R}^{3\times W_{\text{in}}\times H_{\text{in}}}$, which consists of three channels, representing static obstacles, robots, and goal respectively. Note that, $W_{\text{in}} = W_{\text{FOV}} + 2$ and likewise for H_{in}: when the goal is outside the FOV, we mark a point on the edge of the goal channel of $\mathbf{Z}_t^i$ to indicate the direction of the target (Fig. 3.7). This ensures that each robot has only the direction of the target when the goal is outside the FOV.

Network Architecture

CNN-based Perception. Compared to our previous work [19], we upgrade the CNN module with ResNet blocks as follows: we implement a feature extractor with 3 stacked residual blocks. Different from the conventional `Conv2d-BatchNorm2d-ReLU-MaxPool2d` structure, there is a skip connection in each block joining the features before and after the block together. This type of residual connection has been widely used in feature extraction work, and it has been shown to be beneficial to reducing overfitting and improving performance [116].

Graph-based Communication. Each individual robot carries a local copy of the graph convolution layers and communicates its compressed observation vector $\tilde{\mathbf{x}}_t^i$ with neighboring robots within its communication radius r_{COMM}. The resulting fused feature is then passed to the next stage for selecting the action primitive, leading to a localized decision-making scheme.

In the graph convolution architecture, we explore several models with two main types of graph convolution layers: GNN and MAGAT. To better demonstrate the improvements that MAGAT can make in our task, we compare each MAGAT model with the corresponding GNN model with the same configuration except the graph convolution layers. These models are defined in the form of "`[Config_Label]-[Type]-[Num_Features]`":

1. `Config_Label`: can be `GNN` or `MAGAT`. It refers to the graph convolution layer used in this model.

2. `Type`: "`F`" refers to normal CNN-MLP-GraphConvLayer -MLP-Action pipeline, while "`B`" refers to a bottleneck structure (Fig. 3.7), which con-

catenates the feature before the graph convolution layers with those processed features after the graph convolution layers. This bottleneck structure augments the features after communication with the features extracted by the robot itself.

3. `Num_features`: the dimensions of features that engage in communication. In this work, we experiment with 128, 64, 32, and 16.

To demonstrate the information reduction in communication, we constrain the dimensions of extracted features from the CNN module to be 128, and further reduce the dimensionality using an additional MLP layer. The number of input observations F and output observations G are the same and set to `Num_features`. This effectively reduces the dimensions of features that can be shared by the communication network.

In this work, we focus on $L = 1$ (one layer of graph convolution) and $K = 2$ (one-hop communication). Each robot is required to send all its extracted features and receive information from neighboring robots only once. *With the help of our mechanism, the robot is able to direct its attention toward specific communication links, based on the relative importance of the information contained in the messages it is receiving from neighboring robots.*

Action Policy. In the last stage, an MLP followed by a softmax function is used to decode the aggregated features resulting from the communication process into the five motion primitives (up, down, left, right, and idle). During the simulation, the action $\tilde{\mathbf{u}}_t^i$ taken by robot i is predicted by a stochastic action policy based on the probability distribution over motion primitives (weighted sampling). We deployed collision shielding in [19] to ensure collision-free paths. Compared to our previous framework [19], we change the action policy used in the simulations of the validation process from a softmax function (deterministic policy) to weighted sampling (stochastic policy), as using a consistent action policy for validation and test can better indicate which models to select after training.

Training and Online Expert

After training cases are generated and the optimal trajectory of each robot is computed by a centralized controller, the model is trained by the trajectory data, such that it imitates the actions and behaviors of the "expert". We train the model with the pair collection $\mathscr{T} = \{(\mathbf{Z}_t^i, \mathbf{U}_t^{*i})\}_{i=1,\dots,N_{case},t=1,\dots,T_{max}^i}$, where $\mathbf{Z}_t^i$ is the processed observation (Sec. 3.5.3) at time t of case i, while $\mathbf{U}_t^{*i}$ is the expert action at this situation consisting of $\tilde{\mathbf{u}}_t^{*i}$ taken by robot $i = 1,\dots,N_{robot}$. Note that N_{case} and T_{max}^i are the total number of cases and the total steps of

case i, respectively. Thus, our training does not involve time sequence information, requiring the model to learn "instant" reactions based on observations at any given time t.

To further enhance model learning, we deploy the *Online Expert* proposed in our previous work [19] right after every validation process: We select n_{OE} cases randomly from the training set and run the simulation. New solutions are generated for the failed cases using the ECBS solver.

These new solutions become new cases and are appended to the training set: $\mathscr{T}_{new} = \mathscr{T} \cup \{(\mathbf{Z}_t^i, \mathbf{U}_t^{*i})\}_{i=1,\ldots,n_{OE},t=1,\ldots,T_{max}^i}$.

Therefore, our training objective is to obtain a classifier $\mathscr{F}$ with trainable parameters θ given the training dataset $\mathscr{T}$ that is gradually augmented. $\mathscr{L}$ is the loss function.

$$\hat{\theta} = \arg\min_{\theta} \sum_{(\mathbf{Z}_t^i, \mathbf{U}_t^{*i}) \in \mathscr{T}} \mathscr{L}(\mathbf{U}_t^*, \mathscr{F}(\mathbf{Z}_t^i, \mathscr{G}_t(\mathbf{Z}_t^i))) \ . \tag{3.20}$$

Recall that $\mathscr{G}_t$ is the status of the communication network at time t depending on current situation $\mathbf{Z}_t^i$.

3.5.4 Experiments

In this section, we firstly provide details of the experimental setup (Sec. 3.5.4). We then move on to introduce the map sets we use in the experiments (Sec. 3.5.4) and the baselines against which we compare our proposed methods (Sec. 3.5.4). Note that the baseline includes the prior algorithms (HCA and Replanning) and baseline neural network model. Finally, we present and discuss our experimental results (Sec. 3.5.4). For fair comparison, we used the same metrics as in 3.4.2.

Experimental Setup

Our simulations were conducted using the Cambridge High-Performance Computing Wilkes2-GPU NVIDIA P100 cluster with Intel(R) Xeon(R) CPU E5-2650 v4 (2.20GHz). We used the Adam optimizer with momentum 0.9. The learning rate γ was scheduled to decay from 10^{-3} to 10^{-6} within 300 epochs, using cosine annealing. We set the batch size to 64, and L2 regularization to 10^{-5}. The online expert was deployed every $C = 4$ epochs on $n_{\mathrm{OE}} = 500$ randomly selected cases from the training set. Validation was carried out every 4 epochs with 1000 cases that were exclusive of the training/test set.

Table 3.1 Two major map sets with 10% obstacle density for generalization test. Training scenario 20×20 with 10 robots has 30000 cases, partitioned into 70% training set, 15% validation set, and 15% test set. Other scenarios have 1000 testing cases for the generalization test.

Same Robot Density Set		Increasing Robot Density Set	
Map (n_{robot})	Robot Density	Map (n_{robot})	Robot Density
20x20 (10)	0.025	50x50 (10)	0.004
28x28 (20)	0.025	50x50 (20)	0.008
35x35 (30)	0.025	50x50 (30)	0.012
40x40 (40)	0.025	50x50 (40)	0.016
45x45 (50)	0.025	50x50 (50)	0.02
50x50 (60)	0.025	50x50 (60)	0.024
65x65 (100)	0.025	50x50 (100)	0.04

Table 3.2 Large Scale Map Set. The expert computation time in the 3*rd* column is the time cost of using the ECBS solver with an optimality bound 5 to solve the case on a high-performance CPU core. The computation time (successful cases) of our proposed model `MAGAT-B-32-P4` in the same machine is reported in the 4*th* column. Statistics in bold highlight the significant reduction in the computation time of our model.

Map (n_{robot})	ρ_{robot}	Expert time cost (s)	Time cost (s) (std.)
200x200 (500)	0.0125	~3,000 - 4,000	**510.1 (135.0)**
200x200 (1000)	0.025	~33,000. - 35,000	**1286.8 (368.0)**
100x100 (500)	0.05	~1,100 - 1,200	**279.92 (83.0)**

Scenarios

To better compare the generalization capability, we prepare training/valid/test datasets according to Table 3.1. We train and validate our models with 20×20 maps and 10 robots, and then test on all scenarios shown in Table 3.1.

In practice, robot density (ρ_{robot}), simply computed by $\frac{n_{robot}}{W \times H}$, is an effective metric measuring how crowded a scenario is. Note that the obstacle densities in all scenarios in Table 3.1, 3.2 are 10%. We test model generalization ability with two sets of maps. One is "Same Robot Density Set", with all scenarios having the same robot density. Using this map set, we are able to test the generalization of the models on the maps with similar robot density to the training sets. On the other hand, "Increasing Robot Density Set" is a set with increasing robot density, but the map size remains the same. With this map set, we gradually increase the crowdedness, leading to our evaluation of how our model can perform under sparser or more congested environments.

Additionally, we prepare a super-large-scale test set, which has over 500 robots and a very large map size (Table 3.2). There are 50 test cases in each large-scale scenario.

Baselines

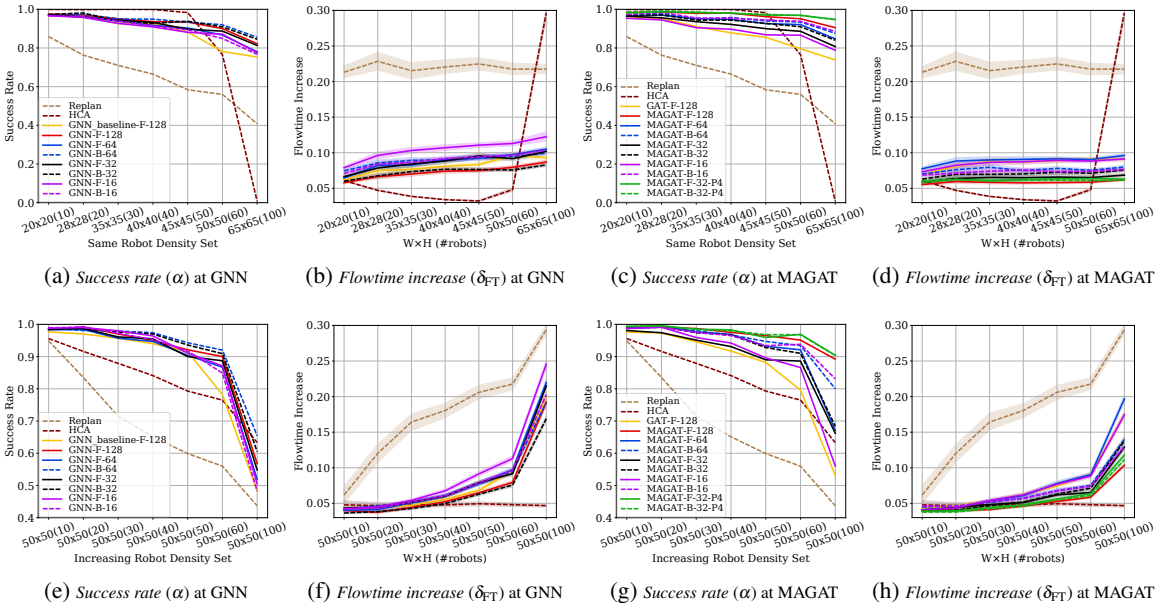
We introduce two models and two non-learning based methods as our baselines.

1. `GNN_baseline-F-128`: The GNN framework we proposed in previous work [19]. Since then, we have upgraded the CNN module for feature extraction and the action policy in the validation process (details in Sec. 3.5.3). Therefore, it represents a good baseline demonstrating the basic improvement that we make to the framework.

2. `GAT-F-128`: The framework is the same as our MAGAT, but instead of our attention mechanism (Eq. 3.10), the mechanism of GAT [107] (introduced in Eq. 3.12) is used.

3. `HCA`: A simple domain abstraction with a heuristic to improve the performance of **centralized** multi-robot path planning [10].

4. `Replan`: Global Re-planning (`Replan`) is an interaction-aware path planning method [2]. If the robot encounters a conflict, the A* method is used to find an alternative path from the current cell to the goal cell, considering all other robots as obstacles.



(a) *Success rate* (α) at GNN (b) *Flowtime increase* (δ_{FT}) at GNN (c) *Success rate* (α) at MAGAT (d) *Flowtime increase* (δ_{FT}) at MAGAT

(e) *Success rate* (α) at GNN (f) *Flowtime increase* (δ_{FT}) at GNN (g) *Success rate* (α) at MAGAT (h) *Flowtime increase* (δ_{FT}) at MAGAT

Fig. 3.8 The *success rate* (α) and *flowtime increase* (δ_{FT}) against the change of environment setup. Here we present the results of GNN models on the left two columns and MAGAT models on the right; we include `HCA` and `Replan` as baselines. The first row is for "Same Robot Density Set", while the second is for "Increasing Robot Density Set". These figures show the effects of reducing bandwidth or using bottleneck structure. In the legend (`[Graph_Layer_Name]-[Type]-[Num_Features]`), `Graph_Layeer_Name` are `GNN` or `MAGAT`, while `Type` - "`F`" and solid line refer to normal CNN-MLP-GNN/MAGAT-MLP-Action pipeline, and "`B`" and dashed line refer to a bottleneck structure. `[Num_Features]` includes 128 (red), 64 (blue), 32 (black), and 16 (purple).

Results

All the results shown in this section are obtained by training on a dataset composed of 20×20 maps with 10 robots (21000 training cases).

Comparison with baselines. As shown in Fig. 3.8, our `GNN-F-128` and `MAGAT-F-128` both outperform their baseline models, `GNN_baseline-F-128` and `GAT-F-128` respectively. For instance, at "Same Robot Density Set", `GNN-F-128` (solid red line) performs slightly better than its baseline `GNN_baseline-F-128` (solid yellow line) when the map size is lower than 40. The gap becomes significant ($> 5\%$ in *success rate*) as we scale the testing set beyond 40×40 with 40 robots. Similar results are observed at "Increasing Robot Density Set", and these demonstrate that our upgrades on the CNN module and the action policy are useful. Under both map sets, `MAGAT-F-128` (solid red line) outperforms `GAT-F-128` (solid yellow line) in both *success rate* and *flowtime increase* significantly, whose performance is only close to that of `MAGAT-F-16` (solid purple line). The underlying reasons for this are discussed in Sec. 3.5.1. `HCA` (light brown) performs quite well with a high success rate and small flowtime increase when cases are simple, but its performance suddenly drops down starting from 40 robots; it cannot find a solution for 100 robots. The success rate (completeness) of `Replan` (dark red) is generally low compared with our methods, with a higher flowtime increase.

Effect of reducing the information shared among robots. We further experiment with limited communication bandwidth, i.e., limiting the size of the shared features from 128 (red), 64 (blue), 32 (black) to 16 (purple), where the full communication bandwidth is set as 128. In Fig. 3.8, we show the *success rate* and *flowtime increase* of GNN and MAGAT family, respectively. With both GNN and MAGAT configurations, the performance will decrease as we reduce the size of shared features. E.g., as the shared features decrease from 128, 64, 32 to 16, the *success rate* of MAGAT models at 65×65 with 100 robots drop from 91%, 85%, 81% to 78% respectively. Yet, the drop of MAGAT is less pronounced than with GNN, where the performance of GNN models decreases from 83%, 78%, 82% to 77% respectively, as the shared features reduce from 128, 64, 32 to 16.

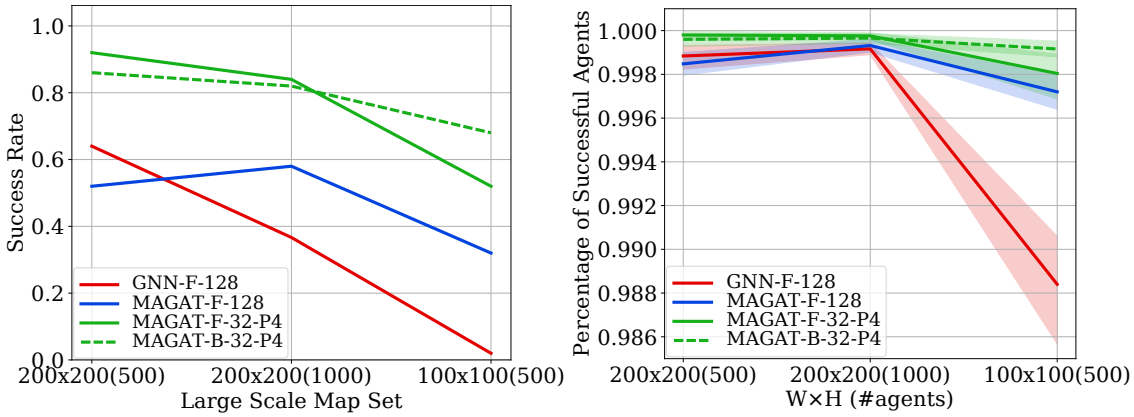
Generalization under different robot densities. Recall that the "Same Robot Density Set" has the same robot density for all its cases. Generalizability is an essential factor to evaluate a machine learning model in real-world applications. The mobility of multi-robot systems naturally leads to time-varying communication topologies. Thus, "Increasing Robot Density Set" allows us to better evaluate the model's flexibility from sparse to more crowded situations. As shown in the Fig. 3.8e, Fig. 3.8f, Fig. 3.8g and Fig. 3.8h, most MAGAT models can maintain the *success rate* above 92% and the *flowtime increase* lower than 10% even as

we increase robot number from 10 to 60 on the same map size (50×50), which demonstrates their good adaptation to different crowdedness.

Effect of bottleneck architecture. Fig. 3.8 also explores the performance of the skip-connected bottleneck structure with GNN and MAGAT. In real-world applications, the communication bandwidth is usually limited which yields the need to have fewer features being shared but performance maintained. For fair comparisons, models are compared to their "baseline" models, for example, comparing `GNN-F-64` (**solid** blue line) and `GNN-B-64` (**dashed** blue line) because they have the same size of features shared across the communication network. With the bottleneck setting, `GNN-B-64` outperforms `GNN-F-64` by around 8% in *success rate* under the scenario of 65×65 maps with 100 robots (Fig. 3.8a). In Fig. 3.8c, even though the performance of MAGAT drops dramatically from 90% (`MAGAT-F-128`, solid red line) to around 78% (`MAGAT-F-16`, solid purple line) in *success rate* at 65×65 maps with 100 robots, the bottleneck structure retains its performance such that it is comparable with `MAGAT-F-128`. This gives the insight that the good generalization ability of MAGAT rests on a sufficient size of the shared features, but it can be preserved by introducing the bottleneck structure. We can conclude that though GNN models do not significantly benefit from a bottleneck structure, this structure helps MAGAT models maintain generalization performance significantly.

Effect of multi-head attention. We also evaluate `MAGAT-F-32` (**solid** blue line) and `MAGAT-B-32` (**dashed** blue line) on their multi-head versions `MAGAT-F-32-P4` (**solid** green line) and `MAGAT-B-32-P4` (**dashed** green line). Note that $P = 4$ parallel MAGAT convolution layers allow larger model capacities, and individual heads can learn to focus on different representation subspaces of the node features [114]. For these two models, as there are $P \times F = 4 \times 32 = 128$ features being shared, the total bandwidth (or the size of shared features) is the same as `MAGAT-F-128`, leading to a fair comparison. In Fig. 3.8c, Fig. 3.8d, Fig. 3.8g and Fig. 3.8h, the multi-head version demonstrates better performance across all the tests regardless of the robot density and map size. Both multi-head models achieve 95% *success rate* in 50×50, 100 robots, and their results for *flowtime increase* under "Same Robot Density Set" remain at low values ($\delta_{\mathrm{FT}} < 0.065$) even with increasing robot numbers.

Super-large-scale generalization. We also test GNN and MAGAT with the large-scale map set, which consists of harder cases that set high requirements for such a model that is trained only on 20×20 with 10 robots. Table 3.2 shows that, with such a large amount of robots (1000 robots), traditional expert algorithms are not capable of solving the problem within an acceptable time. Given the decentralized nature of our framework, the trained models are efficient in the sense that their computation is distributed among the robots, demanding only communication exchanges with neighboring robots. Even though



(a) Success rate

(b) p_{robot}

Fig. 3.9 Generalization test on Large Scale Map Set. a) shows the *success rate*. b) shows the percentage of successful robots ($p_{\text{rg}} = \frac{n_{\text{robots reach goal}}}{n_{\text{robot}}}$), indicating that those model with low *success rates* can still successfully navigated most of the robots to their goals.

`MAGAT-B-32-P4` is only trained by the expert solutions of 20×20 with 10 robots, it can obtain a *success rate* above 80% at 200×200 map with 1000 robots, with only $\frac{1}{30}$ of the computation time taken by the centralized coupled expert (Table 3.2). Fig. 3.9b demonstrated that in all cases, at least 98.6% robots have already reached their goals. At 100×100 map with 500 robots, MAGAT models successfully achieve at least $p_{\text{rg}} = 99.6\%$ robot navigation, but for GNN this number drops down to 98.6%.

Summary. In conclusion, we note that MAGAT has very promising potential in learning a generalizable and flexible dynamic decision-making policy. In general, even in some cases where MAGAT and GNN have a similar *success rate*, MAGAT reduces the flowtime further, leading to more robots achieving their goals and more optimal paths. MAGAT also shows its ability to learn more general knowledge about path finding, which is supported by the fact that it can achieve high performance in large-scale and challenging cases even though it is trained in very simple cases. We also demonstrate that the multi-head attention of MAGAT can improve performance. The trained model achieves a 47% improvement over the benchmark success rate in the 200×200 map with 1000 robots, where the testing instances are $\times 100$ larger than the training instances.

3.6 Conclusion and Future Work

We considered the problem of collision-free navigation in multi-robot systems where the robots are restricted in observation and communication range, and possess no global reference frame for localization. We proposed a combined architecture, composed of a convolutional

neural network that extracts adequate features from local observations, and a graph neural network that communicates these features among robots. The key idea behind our approach is that we jointly trained these two components, enabling the system to best determine what information is relevant for the team of robots as a whole. This approach was complemented by a data aggregation strategy that facilitated the learning process.

This work is the first to apply GNNs to the problem of multi-robot path planning. Our results show that we are very close to achieving the same performance as first-principles-based methods; in particular, we showed our model's capability to generalize to previously unseen cases involving much larger robot teams. Of particular importance is the fact that we can already scale our system to sizes that are intractable for coupled centralized solvers, while remaining computationally feasible through our decentralized approach.

Then, we incorporate an attention mechanism to enable the GNN to selectively aggregate message features. We prove theoretically and empirically that MAGAT is capable of dealing with dynamic communication graphs, and that it demonstrates good generalizability on unseen environment settings and strong scalability on super large-scale cases while we train the model with only very simple cases. We also show that the skip-connected bottleneck structure is a way to maintain model performance while reducing the information being shared. This feature enables MAGAT to achieve good performance while transmitting less data, leading to significant value in practice and applications. The results demonstrate that the multi-head attention is beneficial to MAGAT models, where individual heads can learn to focus on different representation subspaces of the node features [114].

There are some assumptions and corresponding limitations in the current implementation, which will be improved in future work. Firstly, we assumed that communication between robots was achieved instantly without delay. Time-delayed aggregation GNNs [18] can be introduced to extend our framework to handle time-delayed scenarios. Secondly, inter-robot live-locks and position swaps remain a challenge impeding 100% success. One potential solution to this is to deploy a policy gradient to add a penalty on the action causing such scenarios.

Chapter 4

Real World Deployment of Data-driven Policies for Multi-Robot Motion Planning

4.1 Introduction

Significant effort has been invested into finding analytical solutions to multi-robot problems, balancing optimality, completeness, and computational efficiency [11–14]. Data-driven approaches can find near-optimal solutions to NP-hard problems, enabling fast on-line planning and coordination, as typically required in robotics. This has thus provided alternatives for the aforementioned challenges [15, 39, 40, 117]. GNNs, in particular, demonstrate remarkable performance and generalize well to large-scale robotic teams for various tasks such as flocking, navigation, and control [18, 19, 37, 118, 117, 119]. In such multi-robot systems, GNNs learn inter-robot communication strategies using latent messages. Individual robots aggregate these messages from their neighbors to overcome inherently local (partial) knowledge and build a more complete understanding of the world they are operating in.

While GNN-based policies are typically trained in a centralized manner in simulation, and therefore assume synchronous communication, resulting policies can be executed either in a centralized or decentralized mode. Evaluating a GNN in the *centralized* mode typically requires execution on a single machine decoupled from the robots that are acting according to the policy [19, 42, 117]. This *(i)* introduces a single point of failure, *(ii)* requires all robots to maintain constant network connectivity, and *(iii)* introduces scalability issues due to computational complexity $\mathcal{O}(N^2)$ where N is the number of robots. In contrast, in the *decentralized* mode, each robot is responsible for making its own decisions. With fully decentralized evaluation, *(i)* there is no single point of failure, resulting in higher fault tolerance, *(ii)* agents do not need to remain in network range of a router that orchestrates

the evaluation, and *(iii)* computation is parallelized across N robots, decoupling the time complexity from the number of robots.

Even though GNNs have an inherently decentralizable mathematical formulation, previous work on GNN-based multi-robot policies was conducted exclusively in centralized simulations using synchronous communication [19, 118, 18]. For practical reasons, decentralized execution is often unavoidable in the real-world, but it is currently unknown whether this contributes to a shift of domains, and how resulting policies are affected. Multi-robot GNNs require inter-robot communication, but real-world wireless communication is noisy, and messages can be lost or delayed, leading to significant performance loss—this is exemplified in prior work that demonstrates the need for appropriate models to overcome these challenges [120–123]. Further compounding these issues, decentralized policies are typically executed asynchronously, resulting in system states not previously encountered during training.

Extending from our ROS2-based decentralized communication system, we are interested in navigating in an unknown environment to find a target under the guidance of a static *visual sensor network*. Prior work has provided effective solutions employing low-cost wireless sensors to guide robotic navigation [124, 125]. These studies demonstrate that at a small additional cost—i.e., the deployment of cheap static sensors with local communication capabilities—the requirements for the robot's capabilities can be significantly reduced while simultaneously improving its navigation efficiency.

4.2 Related Work

In this section, we first review related multi-robot systems testbeds and frameworks. Our survey includes centralized frameworks as well as decentralized methods that either use machine-learning-based approaches or communication. We emphasize that none of these methods combine learning-based methods and communication. Lastly, we review the related work on robotic communication frameworks and standards to evaluate an appropriate choice for our use case.

Multi-Robot Systems Testbeds Remotely accessible mobile and wireless sensor testbeds are in high demand both in research and industry. Mobile Emulab [126] and CrazySwarm [127] were developed as centrally controlled real-world multi-robot research platforms. As decentralized platforms gained popularity, roboticists developed a variety of systems for small-footprint robot swarms, including Robotarium [128], Micro-UAV [129] or IRIS [130] to large scale platforms such as HoTDeC [131]. These platforms provide testbeds for decentralized control and communication. However, none of these systems utilize machine-learning-based

policies, and only a few learning-based methods have demonstrated real-world experiments [15]. Although work at the intersection of machine-learning and multi-robot control shows remarkable performance [15, 117, 19, 132, 2, 133], little work has been done to show how to make these methods practical (i.e., real-world). Of particular interest is how explicit inter-robot communication [18, 19, 134] plays a role in accumulating information from other robots. A recent study investigates the robustness of decentralized inference of binary classifier GNNs in wireless communication systems [135], but their work is limited to simulation and does not focus on communication contention and latency. These learning-based multi-agent platforms and multi-robot frameworks are either restricted to simulation [117, 19, 2], rely on centralized evaluation [15, 42], or are only evaluated in simulated experiments for decentralized wireless communication. There is a gap between simulation-based testbeds and testbeds that facilitate the deployment of policies derived from machine-learning methods to the real-world.

Robotic Communications Frameworks Communications between agents and controllers is a ubiquitous requirement on experimental robotics platforms, either for experimental control or operational messaging. For these functions, the IEEE 802.11 (commonly WiFi) and 802.15 protocol suites are commonly used [136], with various communications frameworks are overlaid on top of these low-level technologies (e.g. RTPS, MQTT [130] or standard IP [131]). Whatever the specific technology, the underlying protocol suites and the nature of wireless communication set fundamental limitations [137] on available messaging rates when multiple agents are communicating in a decentralized manner. Multiple strategies exist that attempt to maximize protocol performance under specific conditions [138, 139], including dynamic centralization using homogeneous agents [140]. Despite these strategies, the performance of these systems at scale remain poorly tested in real-world robotics systems, which often entail unexpected overheads [141].

Sensor Network-Guided Navigation Most early approaches for sensor network-guided robot navigation assume that either the robot [124, 142] or the sensors [143, 144] are fully positioned in an absolute reference frame, so that an explicit environment map can be created. This information is leveraged to plan the shortest multi-hop route from sensor to sensor, eventually arriving at the target. Shah et al [145] propose a vision-based decentralized controller to aggregate a swarm of agents without relying on inter-agent communication. Methods such as Gaussian Belief Propagation [146] or Factor Graphs [147] use probabilistic models to estimate information such as the position of nodes in a graph using local information iteratively in a computationally efficient manner, which are typically used in robot mapping.

These methods require human prior knowledge to design and tune the model and to extract relevant local features. In contrast, our method is completely data-driven and trained end-to-end, using visual images to directly navigate the robot towards its destination in an unknown environment. Our method promises to scale to any complex real-world scene, where feature extraction for the use of previously mentioned methods can be challenging. Deep Learning (DL)-based methods are becoming more attractive, but approaches such as [148], still require anchor-nodes with access to global positioning information. To the best of our knowledge, there is no related work that uses first-person-view visual observations.

Visual Navigation Learning efficient features from the raw image data is challenging. Hence, auxiliary tasks [149, 150] are used to increase the quality of extracted feature. Curriculum learning [150] is often used for overcoming low sample efficiency and reward sparsity. In contrast to prior work, we consider a novel problem formulation in which the navigating robot is guided by a network of visual sensor that are communicating with the robot and amongst each other. Instead of introducing auxiliary tasks or learning curricula, *we use a joint training scheme to directly learn what information needs to be communicated and how to aggregate the communicated information to ensure efficient navigation in unknown environments.*

GNNs for sensor networks and mobile robot systems. As an effective method to aggregate and learn from relational, non-Euclidean data, GNNs have achieved promising results in numerous domains [151]. In [19, 39], a fully decentralized local motion coordination framework is proposed to solve the multi-robot path-planning problem. In [37], GNNs are used to elicit adversarial communications for self-interested objectives. However, these methods have not considered first-person-view observations. VGAI [152] uses first-person view visual information to imitate a flocking policy in a swarm of drones using GNNs. That problem is different, though, since it is sufficient to extract proximity information from the local robot neighborhood for a reasonable flocking policy. On the other hand, for target navigation, information about the direction to the target has to propagate through the network and reach the navigating robot such that it moves along the shortest collision-free path. Furthermore, no real-world experiments were conducted.

Imitation Learning for Navigation. In its early phases, IL was addressed as a standard supervised learning problem. This approach assumes that data are i.i.d, which is not true as a learned policy influences the future test inputs on which it will be evaluated. This can be alleviated by training over multiple rounds of interaction [153]. DAgger is able to learn

a stationary deterministic policy guaranteed to perform well under its induced distribution of states improves by using reduction-based approaches that enable using supervised learning [112]. AggreVate improves further on this by iteratively estimating a cost-to-go function [112]. Traditional approaches for navigation in unknown environments usually utilize the greedy search with Euclidean heuristic or Manhattan heuristic, which are in-efficient in detecting and escaping local minima [66]. Recent learning-based approaches imitate oracles such as MPC to offer better sample efficiency [154]. In this paper, we employ a visual sensor network to implicitly learn the cost-to-go by predicting the direction corresponding to the shortest path to the target.

Sim-to-Real Transfer and Real-World GNN Implementation. The gap between the simulator and the real world, where dynamics and vision differ, makes it difficult to transfer learned policy. RCANs [155] are proposed to close the gap without using real-world data. Church et al. [156] use image-to-image translation [157] to minimize the difference between real and simulated tactile images. VR-Goggles [158] uses a real-to-sim approach to sim-to-real transfer, where real-world images are first transformed into simulated images, based on a modified CycleGAN [159], so that the policy trained in simulation can be directly deployed. In this paper, we leverage GNNs for robot navigation tasks. Recent work [42] builds the first real-world deployment of GNN-based policies on a decentralized multi-robot platform, but relies on global positioning and a relatively simple state space.

4.3 ROS2-based System for Deploying Decentralized GNN-based Policies

In this section, we developed a framework that facilitates the decentralized execution of GNN-based multi-robot policies. In this work, we initially approach the task in a continuous domain, aligning with the subsequent definition of time presented in the follow-up content. We present the results of a suite of real-robot experiments (see Fig. 4.4) to demonstrate the consequences of this decentralized execution. To that end, we introduce a taxonomy of different evaluation modes and networking configurations. Specifically, we contribute:

1. A ROS2-based software and networking framework for GNNs and other message-passing algorithms to facilitate operation in both simulation and the real-world, and to permit GNN-based policy execution in either a centralized or decentralized manner. We provide the source code online. [1]

[1] github.com/proroklab/ros2_multi_agent_passage

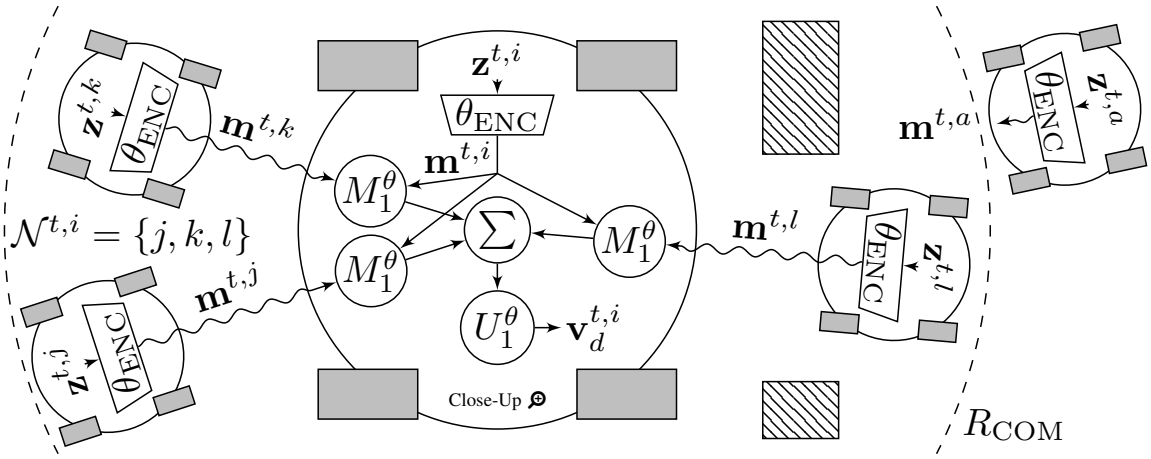


Fig. 4.1 The robots form a graph based on their separation and communication range R_{COM}. They leverage communication via latent messages $\mathbf{m}^{i,t}$ generated from local observations $\mathbf{z}^{i,t}$ propagated over graph edges (wireless Adhoc communication links) to overcome the partial observability of the workspace. To solve this task, we utilize and deploy GNN-based policies that aggregate messages of robots within the local neighborhood $\mathcal{N}^{t,i}$ and compute a local action.

2. An ablation study on several forms of execution to quantify performance shifts between centralized execution and three forms of decentralized policy execution, *(i)* offboard (non-local), *(ii)* onboard over routing infrastructure, and *(iii)* onboard with Adhoc networking.

4.3.1 Preliminaries

In this section, we review the formalization of GNNs as well as the basic functionalities of Robot Operating System (ROS), the software library that we build on.

Graph Neural Networks

A multi-robot system can be defined as graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, where each robot is represented as a node in the node set $\mathcal{V} = \{1, \ldots, n\}$. The inter-robot relationships are represented as edge set $\mathcal{E} = \mathcal{V} \times \mathcal{V}$ with edge features $\mathbf{e}^{t,ji} \in \mathcal{E}$ at each time step t. If robot j is in communication range R_{COM} of robot i, it is in robot i's neighborhood $j \in \mathcal{N}^{t,i}$ and robot i can emit a message $\mathbf{m}^{t,i}$ that is broadcast to its neighbors.

Neural message passing [105] updates the hidden state $\mathbf{h}_k^{t+1,i}$ of each robot i for each neural network layer k using the message function M and the vertex update function U according to

$$\mathbf{h}_k^{t+1,i} = U_k^\theta \left(\mathbf{h}_{k-1}^{t,i}, \sum_{j \in \mathcal{N}^{t,i}} M_k^\theta \left(\mathbf{h}_{k-1}^{t,i}, \mathbf{h}_{k-1}^{t,j}, \mathbf{e}^{t,ji} \right) \right), \tag{4.1}$$

where U and M are functions with learnable parameters θ. The decentralized evaluation is explained in Fig. 4.1. Although centralized formulations also exist, according to (4.1), evaluating a GNN is a fully decentralizable operation depending only on received messages and local information.

ROS and ROS2

ROS is a set of open-source libraries for messaging, device abstraction, and hardware control [160]. ROS generates a peer-to-peer graph of processes (*Nodes*), communicating over edges (*Topics*). ROS requires a master node to connect to all other nodes, preventing its use in fully decentralized systems. ROS2 is a redesign of ROS that solves the master node issue, enabling completely decentralized systems [161]. Many popular frameworks have not migrated from ROS to ROS2, preventing their use in fully decentralized multirobot systems. Our software infrastructure leverages ROS2 to create fully independent agents.

4.3.2 Approach

Our framework can be separated into software and networking infrastructure. In this section, we first explain our software framework. Our framework is capable of running policies in a fully decentralized asynchronous Adhoc mode, but for the purpose of an experimental ablation analysis, we identify a range of sub-categories with different degrees of decentralization.

Specifically, we introduce the four modes: *Centralized* (fully centralized evaluation), Offboard (asynchronous evaluation on a central computer), Onboard o/Infra (decentralization using existing centralized networking infrastructure) and Onboard o/Adhoc (full decentralization using Adhoc communication networks), as visualized in Fig. 4.3. We describe the networking considerations that allow ROS2 to be used for decentralized Adhoc communication between agents.

ROS2 Infrastructure

Our multi-agent ROS2 infrastructure (see Fig. 4.2) allows us to run both simulated and real-world agents concurrently, over multiple episodes, in centralized or decentralized mode, and without human intervention (facilitated through automated resets). An episode is one instance of one experimental trial and a reset is a scenario-specific resetting operation, e.g., requiring robots to move to initial positions. These two actions are repeated for a set number of iterations and different initial states. Our infrastructure follows the Reinforcement Learning (RL) paradigm of delineating the *agent* from the *world*.

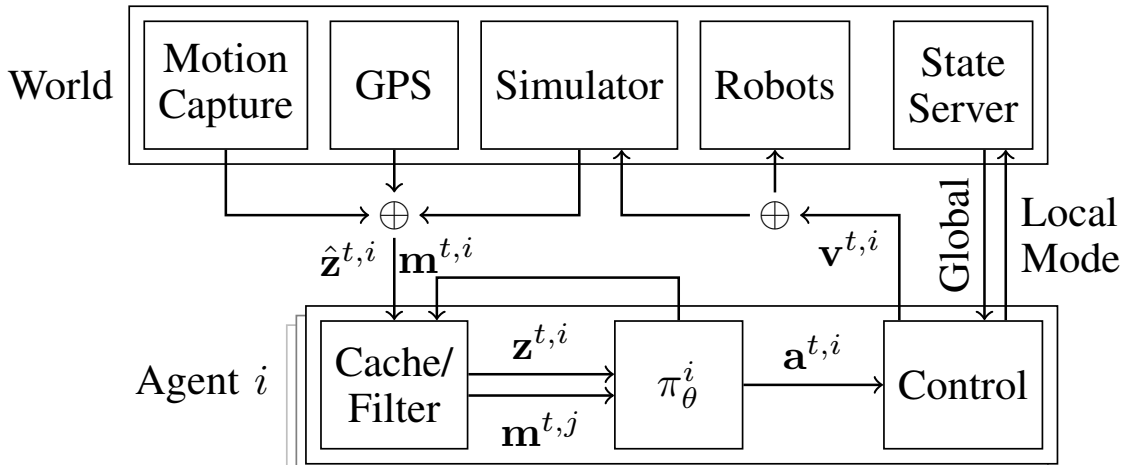


Fig. 4.2 Our ROS2 architecture is composed of the *world* and *agents*. There is one agent $i = 1$ in the centralized case, and multiple $i \in \{1 \dots n\}$ in the decentralized case. The agents receive sensor information $\hat{\mathbf{z}}_t$ from either the motion capture system, GPS, or simulator. The aggregator combines sensor information with messages $\mathbf{m}^{t,i}$ to produce observation $\mathbf{z}^{t,i}$ and neighborhood messages $\mathbf{m}^{t,j}; j \in \mathscr{N}^{t,i}$, for the policy π^i_θ to generate action $\mathbf{a}^{t,i}$. The control node converts the action into velocity commands $\mathbf{v}^{t,i}$. In simulation mode, control drives the simulator instead of the robot wheel motors. The state server orchestrates termination, resets, and operational mode syncs during sequential episodes. This system allows us to run agents in simulation and the real-world concurrently, over multiple episodes, and without any human intervention.

Agent Each agent receives raw sensor data and emits motor commands. The agent is composed of the cache/filter, policy, and control nodes. The cache/filter node uses sensor information $\hat{\mathbf{z}}^{t,i}$ to determine neighboring agents $j \in \mathscr{N}^{t,i}$ within the specified communication radius. It caches neighborhood messages $\mathbf{m}^{t,j}$ and sensor information $\mathbf{z}^{t,i}$ over Δt for the policy. The policy node wraps a trained policy π^i_θ. It receives the observation $\mathbf{z}^{t,i}$ and messages $\mathbf{m}^{t,j}$ and emits a message $\mathbf{m}^{t,i}$ and action $\mathbf{a}^{t,i}$. The action feeds into the control node, which emits motor commands $\mathbf{v}^{t,i}$.

World The world is everything external to the agent. The world can be either *real*, *simulated*, or a mix of both. In the real-world, an external system like GPS or motion capture produces state estimates for the agents. In the simulated world, a rigid-body dynamics simulator receives agent control commands and moves the agents in simulation accordingly. All sim-to-real abstraction is contained within the world, so the agents are unaware if they are operating in the real-world or the dynamics simulator.

The state server is a state machine that coordinates asynchronous episode execution and resets between independent agents. It enables back-to-back episodes and large-scale experimental data collection. It records agent heartbeats, then broadcasts a global operating

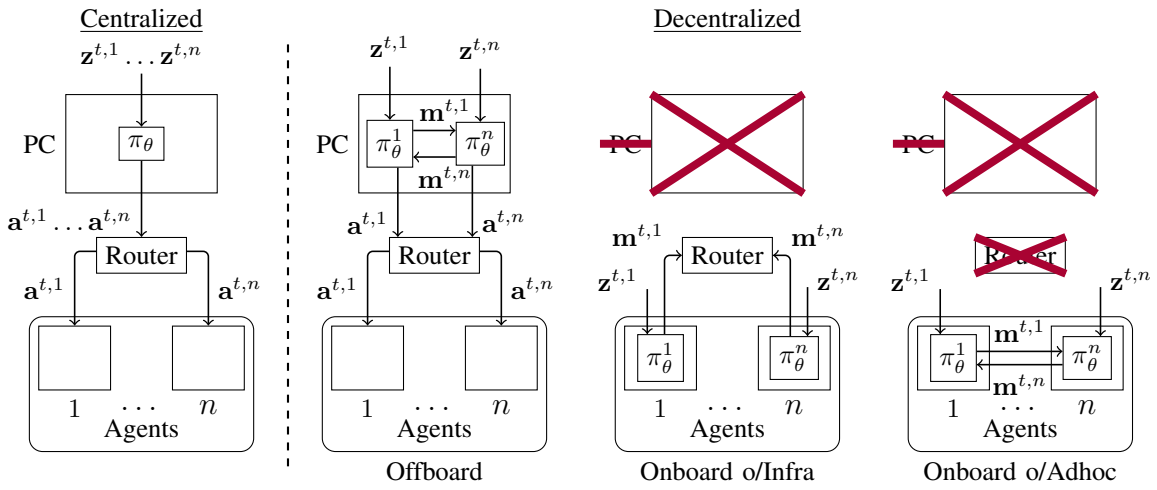


Fig. 4.3 The framework configurations used in our experiments. The ROS2 infrastructure is either centralized or decentralized, with varying degrees of decentralization depending on the network setup. We refer to these four configurations as Centralized, Offboard, Onboard over Infrastructure, and Onboard over Adhoc. Observations $\mathbf{z}^{t,1} \dots \mathbf{z}^{t,n}$ feed into centralized policy π_θ or local policies $\pi_\theta^1 \dots \pi_\theta^n$ to produce actions $\mathbf{a}^{t,1} \dots \mathbf{a}^{t,n}$ for agents $1 \dots n$. Local policies consist of a GNN and pass messages $\mathbf{m}^{t,1} \dots \mathbf{m}^{t,n}$ to communicate. In the centralized case, a single policy produces actions for all agents at once in a synchronized manner. For Offboard, local policies run asynchronously, exchanging messages over `localhost`. The PC is removed for Onboard o/Infra, moving inference onto the robot computers. Onboard o/Adhoc is fully decentralized – the agents forgo the router and communicate directly using Adhoc networking.

mode and initial conditions. Agents use the global operating mode to determine if they should reset or execute the policy.

Communications Networks

Our evaluations consider four different configurations, as summarized in Fig. 4.3, which take the form of variable execution locations (i.e., offboard vs onboard) for policies, and the networks used for messaging between agents and policy execution points. Centralized and Offboard run policies on an external computer, with the remaining two *Onboard* configurations running them on robots' computers. These varied modes allowed us to separate sources of error and performance drop during evaluation.

Our framework uses two wireless communications methods over the various configurations. Both use 802.11, with the first being an *Infrastructure mode* network, and the second being *Adhoc mode*. We selected 802.11 in preference to other Adhoc capable wireless standards, such as IEEE 802.15 due to achievable data rates and compatibility with IP-based networking.

Infrastructure Mode This mode is characterized by a central access point being responsible for managing the network's functions. For Centralized and Offboard configurations only agent actions are sent, which is easily handled by the network. For the Onboard o/Infra configuration, agents forward messages to one another using this message, with observations from the agent location system sharing the network. Finally, in the Onboard o/Adhoc configuration, it handles only the delivery of agent location observations. The implications of each of these modes are discussed further in [42].

Adhoc mode We use this network mode only in the Onboard o/Adhoc mode, where it handles messages between agents. Physically, this network is supported by distinct wireless transceivers carried with each agent, allowing fully decentralized operation. This network takes the form of an 802.11n IBSS, which means that no agent has any special priority access to the wireless medium. Note that this is *not* a mesh network, as there is no facility for multi-hop communications.

ROS2 Middleware Communications with agents exclusively use ROS2 provided middleware for message passing, specifically the eProsima Fast-DDS implementation of RTPS. Due to the fact that we use dynamic agent discovery rather than setting explicit communications routes, an agent-based firewall is deployed to block RTPS messaging traffic from using the incorrect network interface.

4.3.3 Network Infrastructure

We investigate the effects of networking, ROS2, and Fast DDS settings on performance. The evaluation experiment is designed and conducted by our co-author, Jennifer Gielis, which is not the major contribution of the author of this thesis. Therefore, the author will refer the reader to read our paper [42] for the details of network infrastructure.

4.3.4 Case Study: Navigation Through A Narrow Passage

We showcase the capabilities of our framework in a case-study requiring tight coordination between multiple mobile robots. We consider a team of $n = 5$ agents that start in a cross-shaped formation and need to move through a narrow passage to reconfigure on the other side of the wall, as seen in Fig. 4.4. The robots are required to reach their goal positions through collision-free trajectories. Each robot only has knowledge of its own position and goal (i.e., does *not* directly observe the other robots), and is trained to leverage a GNN-based communication strategy to share this local information with neighbors to find the fastest

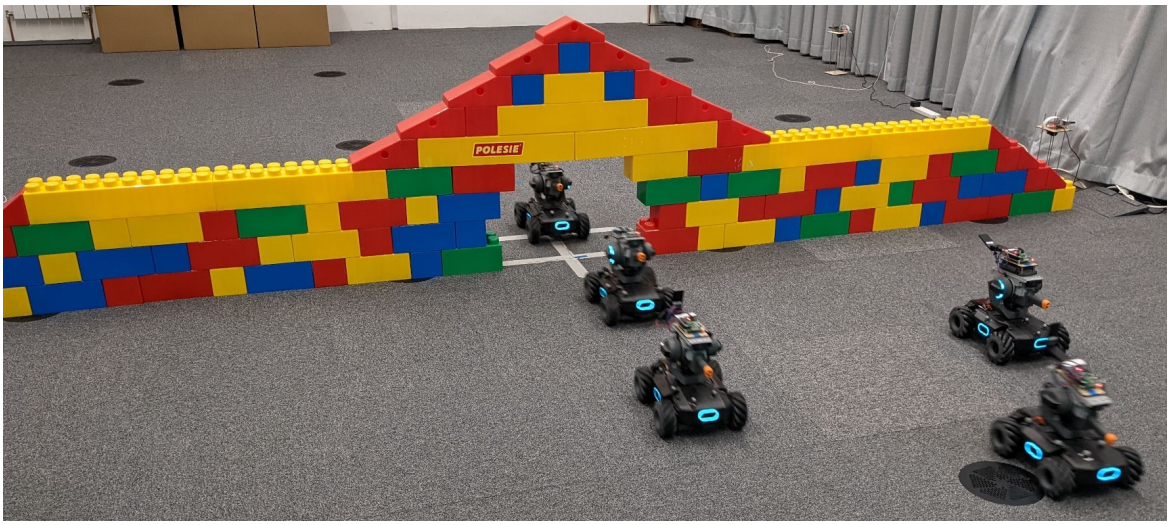


Fig. 4.4 We deploy a set of five DJI RoboMaster robots in a real-world setup using GNNs and Adhoc communication. The robots navigate through a narrow passageway to reconfigure on the other side, as quickly as possible.

collision-free trajectory to its respective goal. An image of this setup can be seen in Fig. 4.4 and a video demonstration is available online[2]. We briefly explain the training and provide the code with implementation details online.[3]

Environment At each time step t, each agent i has a position $\mathbf{p}^{t,i}$, a desired velocity $\mathbf{v}_d^{i,t}$, a measured velocity $\mathbf{v}_m^{i,t}$, and a desired acceleration $\mathbf{a}_d^{i,t}$. We approximate each agent to be circular and implement a simple holonomic motion model that integrates acceleration-constrained velocities into positions. Collisions between agents and the wall result in an immediate stop of the agent. Note that the desired velocity is dictated by the control policy and the measured velocity is current true velocity of the agent. Each agent is assigned a goal position $\mathbf{p}_g^i$. An episode ends if all agents have reached their goal or after the episode times out.

Reward We train agents using RL. The objective of each agent is to reach its goal position $\mathbf{p}_g^i$ as quickly as possible while avoiding collisions. We use a shaped reward that guides individual agents to their respective goal positions as quickly as possible while penalizing collisions.

Observation and Action The observation $\mathbf{z}^{i,t}$ consists of locally available information, specifically the absolute position $\mathbf{p}^{i,t}$, the relative goal position $\mathbf{p}_g^i - \mathbf{p}^{i,t}$, as well as a predicted

[2]youtube.com/watch?v=COh-WLn4iO4
[3]github.com/proroklab/rl_multi_agent_passage

position $\mathbf{p}^{t,i} + \mathbf{v}^{i,t}$. The desired velocity is the policy's action output $\mathbf{v}_d^{i,t} = \mathbf{a}^{i,t}$. We constrain acceleration and velocity to $a_{\max} = 1\text{m/s}^2$ and $v_{\max} = 1.5\text{m/s}$.

Model As the model for the policy π_θ^i we use the GNN introduced in Sec. 4.3.1. The number of layers is constrained by our communication framework. Since more layers result in multiple rounds of communication exchanges at the same time step, we set $k = 1$. Each message is an encoding of the observation so that $\mathbf{m}^{t,i} = \mathbf{h}_0^{t,i} = \theta_{\text{ENC}}(\mathbf{z}^{t,i})$. We define our message function and vertex update function as $M_1^\theta(\mathbf{h}_0^{t,i}, \mathbf{h}_0^{t,j}, \cdot) = \theta_{\text{GNN}}(\mathbf{h}_0^{t,i} - \mathbf{h}_0^{t,j})$ and $U_1^\theta(\cdot, x) = \theta_{\text{ACT}}(x)$. Furthermore, we include self-loops and thus consider agent i as part of its own neighborhood so that $\mathcal{N}^{t,i} = \mathcal{N}^{t,i} \cup \{i\}$. The output of the GNN is the desired velocity $\mathbf{v}_d^{t,i} = \mathbf{a}^{t,i} = \mathbf{h}_1^{t,i}$. θ_{ENC}, θ_{GNN} and θ_{ACT} are learnable Multilayer Perceptrons (MLPs). We use the same approach as described in [37] to train our model using PPO with local rewards for each agent.

Experimental Setup In total, we run a series of six different real-world experiments for the four modes (Fig. 4.3) to demonstrate the capabilities and performance of our framework and two additional experiments to demonstrate the robustness of our policy against changes to the communication radius in the real-world. In addition to using a set communication radius of $R_{\text{COM}} = 2\text{m}$, we *(i)* run the policy in a fully connected communication topology, and *(ii)* run the policy in a noisy communication topology by modeling the communication range as a Gaussian with a mean of $R_{\text{COM}} = 2$ m and a standard deviation of 0.5 m (the policy is trained with $R_{\text{COM}} = 2$ m).

To collect a statistically significant amount of data, we generate $E = 16$ episodes, each with a different set of random start and goal positions, and repeat each episode for each experiment $K = 12$ times, resulting in $K \cdot E$ episodes in the training environment (simulation) and on real robots. We use customized DJI RoboMaster robots equipped with Raspberry Pi's that locally run policies. The robots are provided with state information as explained in Sec. 4.3.2.

Results

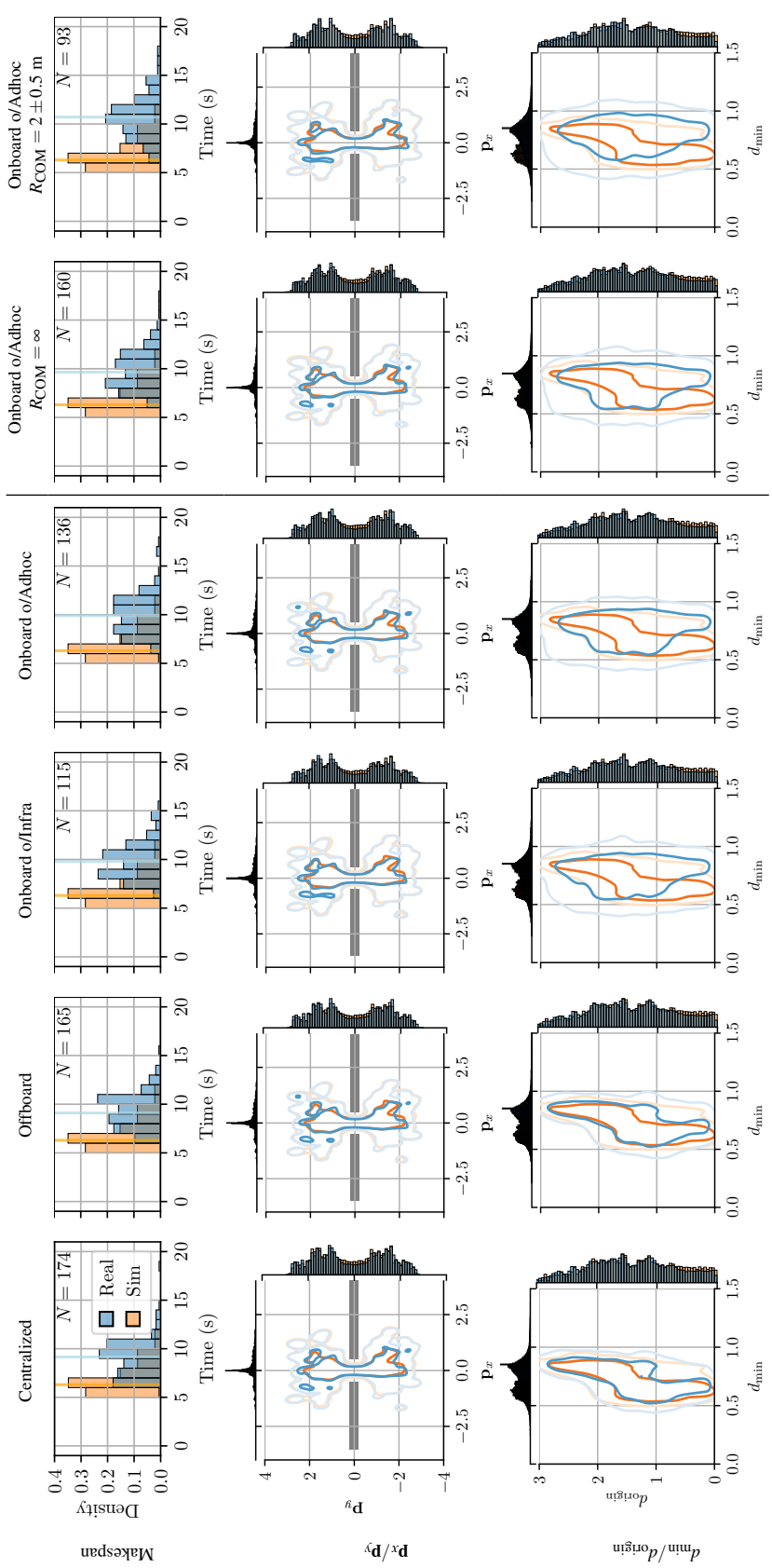


Fig. 4.5 We visualize a variety of makespan and position distributions over the six experiments we conducted. The columns show the data of the centralized simulation baseline in orange and the data of the corresponding real-world experiment as labeled in the column headers in blue. For each experiment, we run a total of 192 episodes with 16 different start and goal positions. The last column compares the Onboard o/Adhoc experiment with a simulation evaluated with communication delays. The first row shows the probability densities of makespans of successful episodes (episodes that did not result in a collision with the wall and for which all robots reached their goal, indicated with N). The median makespan is indicated with a dashed line. The second row shows the distribution of positions, indicating the position of the wall and the passage. The third row shows the distribution of minimum distances between robots at each time step d_{min} and distance from the origin or passage d_{origin}.

Table 4.1 Overview of performance metrics for all case study experiments.

	simulation	Centralized	Offboard	Onboard o/Infra	Onboard o/Adhoc	Onboard o/Adhoc $R_{\text{COM}} = \infty$	Onboard o/Adhoc Noise
Success Rate	95.8%	90.1%	85.9%	59.4%	70.3%	66.7%	62.5%
Median Makespan	6.3 s	9.1 s	9.1 s	9.8 s	9.9 s	9.6 s	10.7 s

We use two metrics to evaluate the performance of our model in simulation and real-world. The success rate is the fraction of collision-free episodes for which all robots reached their goal. The makespan is the time it takes for the last agent to reach its goal. For both metrics, episodes with wall or inter-agent collisions are excluded. Inter-agent collisions are defined as two agents approaching each other closer than 0.32 m. We compare it to a simulation baseline, for which the policy is evaluated during training conditions. We show distributions of makespans and positions in Fig. 4.5 and show quantitative results in Tab. 4.1.

The Centralized case reflects the performance gap caused by dynamic constraints that are not considered in simulation. Since the GNN is evaluated synchronously, communication is not affected by real-world effects. The makespan is about 1.5 times worse and the success rate 5.7 percentage points (pp) worse than in simulation.

The Offboard mode evaluates the GNN asynchronously across different processes on the same physical computer. Compared to the Centralized mode, it features *asynchronous* evaluation but little to no inter-process communication delays, resulting in slightly worse performance of 4.2 pp and worse median makespan of 0.2s wrt the Centralized mode.

The Onboard o/Infra mode moves the decentralized GNN from a central computer to the on-board computers of each individual robot and therefore adds communication delays caused by wireless routing and contention. We notice a decrease in performance of 26.5 pp in terms of success, and a deterioration of 1.0 s of median makespan w.r.t. the Offboard mode. Onboard o/Adhoc mode improves the performance by 10.9 pp, with a similar median makespan. This can be attributed to less contention.

Setting $R_{\text{COM}} = \infty$ results in an identical median makespan and a slight decrease in performance of 3.6 pp. This decrease is expected due to out-of-distribution neighborhoods that never occur during training (while the agents are typically fully connected in the start and the beginning of each episode, they are not when moving through the passage). When adding noise to the communication range, the success rate drops by another 4.2 pp (or 7.8 pp wrt the Onboard o/Infra mode) and 0.9 s median makespan.

The second and third row in Fig. 4.5 visualize distributions over positions. The second row shows that the distribution of absolute positions over all experiments are consistent, even when comparing to the centralized simulation. In the third row, we compare the distribution of distance to the origin (or the passage) d_{origin} over minimum distance between agents d_{min}.

While the simulation and real distributions are overlapping in the Centralized and Offboard mode, there is a noticeable discrepancy in all Onboard modes, especially for small d_{min}, for which d_{origin} is shifted towards higher values, indicating that the robots are further away from each other when close to the passage, which can be attributed towards slower reaction times caused by communication delays.

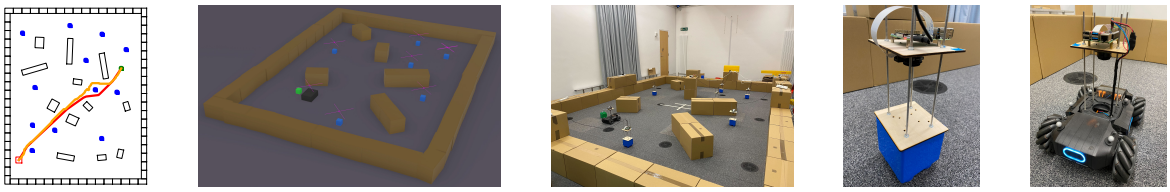
We run an additional simulation that evaluates the GNN in a decentralized mode with communication delays. We observed that for higher delays, the success rates dropped significantly, while the makespan decreased much less notably. The distribution of d_{min} over d_{origin} shifted slightly towards the real-world distribution. This indicates that the shift in makespan we observe is mostly due to robot dynamics, and real-world communication latency causes the agents to be less responsive and therefore to collide.

4.4 Learning to Navigate using Visual Sensor Networks

The implementation of traditional sensor network-guided navigation may be cumbersome. It commonly consists of five main steps: *(1)* estimate robot and sensor positions through external positioning systems, *(2)* process sensor data to detect the target, *(3)* transmit target information to the robot, *(4)* build the environmental map and plan a target path, and *(5)* control the robot to follow the path according to its motion model. This framework has several drawbacks. First, parameters need to be hand-tuned, and data pre-processing steps are required. Second, isolating the perception, planning, and control modules hinder potential positive feedback among them, and make the modeling and control problems challenging. Finally, this approach requires the availability of an absolute positioning system, to build on step (1).

In this section, we explore a learning-based approach where we introduce a static visual sensor network that can acquire the ability to guide a robot toward its intended destination. The nodes in this sensor network are endowed with policies that are learned through a machine learning architecture that leverages a Graph Neural Network (GNN). Successful navigation requires the robot to learn the relationship between its surrounding environment, raw sensor data, and its actions. In this work, we initially approach the task in a continuous domain, aligning with the subsequent definition of time presented in the follow-up content. Our contributions are as follows:

- We present a framework that demonstrates, for the first time, how low-cost sensor networks can help robots navigate to targets in unknown environments, *where neither the robot nor the sensors possess any absolute positioning information*.



(a) Map (b) Webots (c) Real world (d) Sensor (e) Robot

Fig. 4.6 Our setup consists of (a) an environment populated with obstacles, visual sensors (blue), a mobile robot equipped with a sensor, and a green target, where the mobile robot has to find the shortest path (red, taken path orange) to an occluded target by incorporating communicated sensor information to enhance its navigation performance. Our framework leverages a setup consisting of a simulation environment (b) that corresponds to the real-world setup (c). The workspace is endowed with custom-built sensors with fish-eye cameras (d, e) that are capable of communicating with each other. One sensor is attached to a mobile robot (e) that acts on the output of that sensor.

- We provide an end-to-end visual navigation policy that leverages GNNs to *learn what needs to be communicated* (among sensors and robot) and how to aggregate the visual scene for effective navigation.
- Experimental results demonstrate generalizability to unseen environments with various sensor layouts. In particular, by introducing a *real-to-sim image translator*, our policy (which is trained entirely in simulation) can be transferred to the real world without additional tuning (i.e., in a zero-shot manner).

4.4.1 Problem Formulation

Our setup consists of a cluttered environment, a mobile robot, and a set of visual sensors. The environment $\mathscr{W}$ contains a set of randomly placed static obstacles $\mathscr{C} \subset \mathscr{W}$ and N randomly placed visual sensors $\mathscr{S} = \{S^1, \ldots, S^N\}$. As shown in Fig. 4.6 (c), at every time step t, each sensor S^i is capable of taking an omnidirectional RGB image o_t^i of its surrounding environment, but has *no positioning information*. Each sensor S^i can communicate with nearby sensors within communication range, i.e., $S^j \in \mathscr{N}^i$. A target object G is located randomly in the 2D ground plane at position q^G. Each sensor predicts the direction $u_t^i \in \mathscr{U}$ along the shortest path towards the target G. The mobile robot R is located at position q_t^R and moves in the ground plane in $\mathscr{W} \setminus \mathscr{C}$. It is equipped with *any one* of the sensors (we choose sensor S^1) and uses the directional output u_t^1 of that sensor to execute an action $a_t \in \mathscr{A}$ by applying a velocity of the same direction. *The robot's objective is to move to the target G along the shortest collision-free path. It utilizes the information shared through the sensor*

network to make an informed decision about how to reach the (potentially occluded) target, while avoiding time-consuming exploration.

We formalize this as a sequential decision making problem under uncertainty about the underlying world [66], and define a corresponding Markov Decision Process (MDP). At time step t, let $s_t \in \mathscr{O}$ be the observed state of the environment, i.e $s_t = \{o_t^1, \ldots, o_t^N\}$. On executing a_t, the new state s_{t+1} is determined by the underlying world $\mathscr{W}$, which is a hidden variable, sampled from a prior $P(\mathscr{W})$ and in turn induces a state transition distribution $P(s_{t+1}|s_t, a_t)$. The one-step cost $c(s_t, a_t)$ is the distance travelled since the previous time step and a consequence of action a_t.

Let $\pi(s_t)$ be a policy that maps the state s_t to an action a_t. The policy represents the navigation strategy that we wish to learn. An episode continues until either the goal is reached ($\|q^G - q_t^R\| < D_G$) or continuous time horizon T is reached. Given a prior distribution over worlds $P(\mathscr{W})$ and a distribution over start and goal positions $P(q_0^R, q^G)$, we can estimate the cost of moving from position q_0^R to q^G as

$$V(s_t) = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_\pi^t} [c(s_t, \pi(s_t))] \tag{4.2}$$

where $d_\pi^t = P(s_t|\pi, \mathscr{W}, q_t, q^G)$ is the distribution over states induced by running π on the problem $(\mathscr{W}, q_t, q^G)$ for T steps [112], and we evaluate the performance of a policy as

$$J(\pi) = \mathbb{E}_{\substack{\mathscr{W} \sim P(\mathscr{W}), \\ (q_0^R, q^G) \sim P(q_0^R, q^G)}} [V(s_t)]. \tag{4.3}$$

Assumptions. The robot has no knowledge of its own position, nor the environment map, nor the target location. The target has to be within visibility range at least one sensor within multi-hop communication to the robot, but it is not necessary for the robot itself to observe the target. In order for the sensors to be able to localize themselves within the environment and with respect to the robot, a minimum overlapping field of view is required. We do not assume any ordering or identification of sensor nodes. Our approach is stateless, and we do not use memory to store information over multiple time steps (i.e., we do not build a map).

4.4.2 Visual Navigation using Sensor Networks

This section describes our approach to training the navigation policy π. Fig. 4.6 shows the simulation environment and Fig. 4.7 shows an overview of our architecture. The objective of each sensor S^i is to predict a direction u_t^i along the shortest path to the target (with the consideration of static obstacles) by using its own observation o_t^i and the messages shared by

other sensors. We use a variant of Imitation Learning (IL) using expert solvers (i.e., cheap and fast path planning algorithms).

Our method does not rely on any positioning information whatsoever, and as explained in Sec. 4.4.1, the neighborhood $\mathscr{N}^i$ is defined through sensors within communication range. During training and evaluation in simulation, we have to model this communication range. We assume a disk model, where the neighbor set is defined as $\mathscr{N}^i = \{S^j | L(S^i, S^j) \leq D_S\}$, where $L(S^i, S^j)$ is the euclidean distance between S^i and S^j, and D_S is the communication range.

In Sec. 4.4.2, we explain how we generate the dataset $\mathscr{D}_{\text{IL}} = \{(o_0^{i,m}, \hat{A}_0^{i,m})\}_{m=1,i=1}^{M,N}$ consisting of M per-sensor cost-to-go advantage labels $\hat{A}$ and observations o. In Sec. 4.4.2, we detail the neural network model $\psi \circ \theta \circ \phi(\cdot)$ consisting of the feature extractor ϕ, the feature aggregator θ and the post-processor ψ to predict the advantages. The architecture is depicted in Fig. 4.7. This neural network is optimized with the objective J_{IL} as

$$J_{\text{IL}}(\psi \circ \theta \circ \phi) = \mathbb{E}_{o,\hat{A} \sim \mathscr{D}_{\text{IL}}} \left[\| \psi \circ \theta \circ \phi(o) - \hat{A} \|^2 \right]. \tag{4.4}$$

While we train our policy in simulation, we take an approach that facilitates sim-to-real transfer (see Sec. 4.4.3).

Data Generation

Imitation learning can be treated as supervised learning problem. Since future states are influenced by previous actions in IL, the i.i.d. assumption of supervised learning is invalidated. AGGREVATE [112] proposes one possible solution to this problem by iteratively learning cost-to-go estimates for trajectories of data using an online procedure. For the type of navigation policy that we wish to learn, it is trivial to compute cost-to-go estimates, and therefore we treat it as standard supervised learning problem with an i.i.d. assumption for each sample.

Let Q^i be the cost-to-goal for sensor S^i. Let the set of possible directions be $\mathscr{U} = \{u^1, \ldots, u^K\}$. Let Q^i_{uk} be the cost-to-goal for sensor S^i upon moving towards direction u^i_k. To simplify the training procedure and facilitate better generalization, instead of learning absolute cost-to-goal values Q^i, we learn a relative cost-advantage vector $A^i = [Q^i_{a1} - Q^i, \ldots, Q^i_{ak} - Q^i]^\intercal$ that subtracts the cost from the current state to the goal for each of the K possible directions and thus is scale-invariant (since the scale of the cost-to-go values are not relevant for a sequential decision making problem).

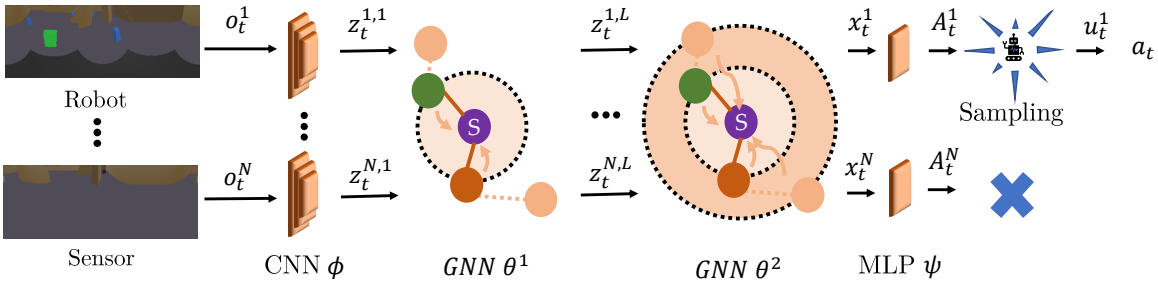


Fig. 4.7 We train a policy in simulation using simulated images o_t^i. The CNN $\phi(\cdot)$ encodes images into features z_t^i and the GNN $\theta(\cdot)$ further encodes this for multiple communication hops in L layers as $z_t^{i,l}$. Eventually, the post-processing MLP $\psi(\cdot)$ generates cost-to-go advantages A_t^i which are used to sample a direction to the target along the shortest path as u_t^i and eventually to generate action a_t for the robot, which is equipped with sensor S_1.

We generate the dataset $\mathscr{D}_{\text{IL}}$ consisting of $M = 40,000$ environment samples for $N = 7$ sensors for training the policy π. We provide further details on the dataset and the training in Sec. B.1.

GNN-based Feature Aggregation across Sensor Network

The neural network is homogeneous across all sensors (and robot) and is divided into the three sub-modules feature extractor $\phi(\cdot)$, feature aggregator $\theta(\cdot)$ and post-processor $\psi(\cdot)$.

Local feature extraction. We first use a Convolutional Neural Network (CNN) $\phi(\cdot)$, specifically MobileNet v2 [162], to extract features z_t^i from the image o_t^i of each sensor S^i. MobileNet is optimized for the evaluation on mobile devices, which makes it an ideal candidate for the application in a distributed sensor network. For the sim-to-real transfer, we replace the encoder, as explained in detail in Sec. 4.4.3 and visualized in Fig. 4.7. The encoding is communicated to other sensors within communication range.

Neighborhood feature aggregation. In order to predict the target direction, our models need to be able to aggregate information across the whole sensor network. In other words, each sensor requires effective information from those sensors that can *directly* see the target. This feature aggregation task is more challenging than the traditional GNN-based feature aggregation for information prediction [151] or robot coordination tasks [39, 152]. Specifically, in the aforementioned papers, each agent only needs to aggregate information from the nearest few neighbors as their tasks can be achieved by only considering local information. For each agent, information contributed by a very remote agent towards improving the

prediction performance can vanish as the network becomes larger. Additionally, in our task, only a limited number of sensors can directly 'see' the target. *Yet, crucially, information about the target from these sensors should be transmitted to the whole network, thus enabling all the sensors to predict the target direction from their own location (which is potentially in NLOS*). In addition, as we do not introduce any global nor relative pose information, in order to predict the target direction, each sensor must implicitly learn the ability to estimate the relative pose to its neighbors by aggregating image features. Furthermore, generating an obstacle-free path in the target direction by only using image features (without knowing the map) is also very challenging.

The multi-layer GNN model $\theta^{(L)}(\cdot)$ consists of L layers and generates a new encoding $z_t^{i,l}$ for each layer L. It takes the image encoding generated by the encoder $z_t^{i,1} = z_t^i$ as input to the first layer, aggregates communicated neighbors' features using the neighborhood $\mathscr{N}^i$ over multiple layers, and extracts fused features $x_t^i = z_t^{i,L}$ where $z_t^{i,L}$ is the encoding generated by the last GNN layer for each sensor S^i so that $\theta^{(L)}(z_t^i) = x_t^i$.

A GNN model consists of a stack of neural network layers, where each layer aggregates local neighborhood information, i.e., features of neighbors, around each node and then passes this aggregated information on to the next layer. Specifically, our method builds on the GNN layer introduced in [163]. We use a recursive definition for a multi-layer GNN where each layer $L > 0$ computes new features as

$$\theta^{(L)}(z_t^{i,L}) = \sigma\Big(\theta^{(L-1)}(z_t^{i,L}) \cdot W_1^{(L)} + \sum_{j \in \mathscr{N}^i} \theta^{(L-1)}(z_t^{j,L}) \cdot W_2^{(L)}\Big) \tag{4.5}$$

that are communicated in the neighborhood $\mathscr{N}^i$ where $W_1^{(L)}$ and $W_2^{(L)}$ are trainable parameter matrices, and σ denotes a component-wise non-linear function, e.g., a sigmoid or a ReLU.

Cost-to-goal prediction. Lastly, we utilize a post-processing Multi-Layer Perceptron (MLP) to predict a set of cost-to-go advantages for each sensor so that $\psi(x_t^i) = A_t^i$, which is eventually used to sample a target direction u_t^i and eventually transform it to an action a_t.

Policy. This results in the composition $\psi \circ \theta \circ \phi(o_t^i) = A_t^i$ to compute a cost-to-go advantages from the local image o_t^i and features $z_t^{i,l}$ communicated through the neighborhood $\mathscr{N}^i$. We model the set of target directions $\mathscr{U}$ as a discrete distribution so that $u_t^i = \text{cat}(\text{softmax}(-\alpha A_t^i))$ where $\text{cat}(\cdot)$ samples from a categorical distribution and α is a hyperparameter to adjust the stochasticity. As explained in Sec. 4.4.1, the relationship between direction u and action a is bijective. Hence, any sensor can be used as part of a mobile robot (to command its motion). In this work, we denote that sensor as S_1. So far, we

have used a distributed notation to compute a direction u_t^i for any given sensor. We formulate the target navigation problem as centralized MDP relying on a central state $s_t = \{o_t^1, \ldots, o_t^N\}$ consisting of the observation of all sensors. Therefore, we can define the policy π as

$$\pi(\{o_t^1, \ldots, o_t^N\}) = \text{cat}(\text{softmax}(-\alpha(\psi \circ \theta \circ \phi(o_t^1))))\beta = u_t^1 \beta = a_t, \tag{4.6}$$

where β is a hyperparameter to scale the magnitude and thus transforming a direction into a velocity action $a_t \in \mathscr{A}$. Note that even though the notation of the policy is centralized, the execution is inherently decentralized, and instead of depending on raw image observations for other sensor nodes, the robot only depends on the local observation o_t^1 and latent encodings communicated through the GNN θ. We provide more details on the neural network architecture in Sec. B.1.

4.4.3 Zero-Shot Real World Transfer

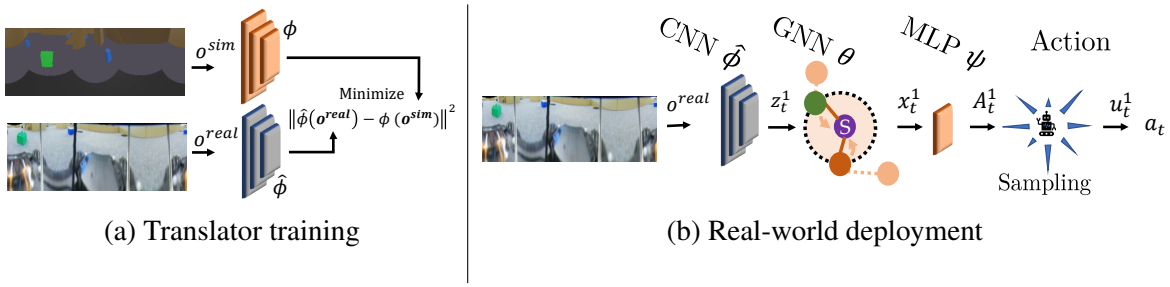
To demonstrate the feasibility of a zero-shot transfer[4] of our model, we design a twin environment setup, consisting of both real and digital copies of the operational space, see Fig. 4.6. The real setup includes custom-built sensors that provide local visual sensing to nearby nodes within the communication network. Later, in Sec. 4.4.4, we report results that demonstrate the effectiveness our sim-to-real approach for the real-world scenario. We provide more details on the real-world transfer in Sec. B.1.

Setup

Twin Environments. The real-world environment is 5.7 m $\times$ 4.2 m, and is cluttered with three to five obstacles of different size. We use standard cardboard boxes for obstacles and blue and green building blocks to identify sensor locations and target location, respectively. In order to facilitate zero-shot transfer, we design a digital twin of our real-world environment. This digital twin (i.e., a simulation environment) is built within the Webots simulator [165] and is illustrated in Fig. 4.6.

Custom-made sensor nodes. We design and construct six sensor nodes that consist of a contraption holding the downward-facing camera with fisheye lens as well as a local data processing unit. The sensor nodes are equipped with a Raspberry Pi running local processing and image reprojection according to a custom camera calibration procedure, and streaming

[4]Zero-shot learning is a feature of a model to be able to adapt to new task without the need for additional learning/fine-tuning [164].



(a) Translator training (b) Real-world deployment

Fig. 4.8 Our sim-to-real framework. (a) After training the policy in simulation, we collect image pairs from simulation o^{sim} and the real world o^{real} and use them to train a real-to-sim translator model $\hat{\phi}$ (gray) by reconstructing latent features from the simulation domain z generated through the encoder trained in simulation ϕ (orange) with real-world images. (b) We combine the translator model $\hat{\phi}$ trained on real-world images (gray) with θ and ψ trained in simulation (orange) to deploy the policy to a real-world setup.

of image data at a frame rate of 12 Hz. The sensor was designed so that it can be used stand-alone, as well as mounted on a mobile robot.

Mobile robot. We use the DJI RoboMaster as mobile robot platform. A seventh sensor node that also serves as controller for the robot is mounted on top. During navigation, the robot employs a collision shielding mechanism, that takes as input distance measurements to detect the near-sided static obstacles and the border of environment. Repulsive force against detected obstacles are generated through a potential field [166]. This mechanism only triggers for near collisions and is a safety mechanism.

Domain Adaptation

As outlined in Fig. 4.7, we first train the policy using simulated images through IL. Fig. 4.8 outlines how we perform the sim-to-real transfer. We create a real-world environment and map it using a motion capture system. We transfer this map into Webots to have an identical representation of the environment in the real world and in simulation. To train the translator, we tele-operate the robot in this environment to collect M image pairs of real-world images o_{real} and corresponding simulated images o_{sim} and store them in a dataset $\mathscr{D}_{\text{transfer}} = \{(o^j_{\text{real}}, o^j_{\text{sim}})\}_{m=0}^{M}$.

In total, we construct 8 different environments, each populated with 6 sensors and one robot. We collect 2000 image pairs for each sensor in each environment (this is achieved within 5 minutes with images being recorded at 10 Hz). Constructing and mapping each environment and setting up the data collection procedure takes 30 minutes per environment. We automatically filter the images in post processing for the sensor images to only be included

when the robot is moving within the field of view of the sensors. This results in a dataset of $M = 80,000$ image pairs for the training set and $10,000$ images for the test set.

We use a similar approach to [158], where a neural network based translator model is trained to map real-world images to simulated images, which are then fed to the policy, but instead of mapping to simulated images, we map to their respective encoding. Specifically, we train a translator model $\hat{\phi}(\cdot)$ that maps real images o^{real} to the encoding of the corresponding simulated image $\phi(o^{\text{sim}})$ by minimizing the objective $J_{\text{transfer}}(\cdot)$ while keeping $\phi(\cdot)$ fixed,

$$J_{\text{transfer}}(\hat{\phi}) = \mathbb{E}_{o^{\text{sim}},o^{\text{real}}\sim\mathscr{D}_{\text{transfer}}}\left[\|\hat{\phi}(o^{\text{real}}) - \phi(o^{\text{sim}})\|^2\right]. \tag{4.7}$$

4.4.4 Results

We first introduce the metrics we use for evaluation and then demonstrate the performance of our method in simulation, based on the methodology introduced in Sec. 4.4.2. Second, we demonstrate successful zero-shot transfer to the real-world setup introduced in Sec. 4.4.3. We provide additional results in Sec. B.1.

Performance Metrics

We evaluate the trained policies on the unseen test split of the training dataset as well as a generalization set that has been generated with a larger environment and more sensors.

We consider two primary metrics for our evaluation. A run is considered successful if the robot arrives at the target without collisions and within a pre-defined time horizon T, captured by the boolean success indicator C_m for case m. The boolean *success rate* fraction of all successful runs is $\frac{1}{M}\sum_{m=1}^{M} C_m$. We furthermore report the success weighted by path length, or SPL [167], as $\frac{1}{M}\sum_{m=1}^{M} C_m \frac{P_m}{\max(p_m,P_m)}$, where P_m is the shortest path length from the robot's initial position to the target, and p_m the length of the path actually taken. We visualize a selection of paths in Fig. 4.10. We report all results for environment configurations where the target is within line-of-sight (LOS) initially (therefore trivially solvable without communication) and where the target is in non-line-of-sight (NLOS, therefore requiring additional sensor coverage for an efficient solution) separately.

Simulations

We train five variants of our policy to evaluate our approach. We first train a policy with communication range $D_S = 0.0$ as a baseline (i.e., no communication). We also train three policies with communication ranges of $D_S = 2.0$, $D_S = 4.0$ and $D_S = \infty$ (fully connected) respectively, all with a single GNN layer. Lastly, we train a policy with a communication range of $D_S = 2.0$ and two GNN layers. The results can be seen in Tab. 4.2.

We evaluate all policies on $M = 250$ unseen maps on the small test set and $M = 100$ maps on the large test set. All policies have a nearly perfect success rate of close to 1.0 for LOS, which is to be expected as navigating to the target using local information is trivial. The baseline policy without communication has an NLOS success rate of 0.827 and an SPL of 0.719 in the small environment. These values are the lower bound for what is possible without communication. We ensure that all environments are solvable. Even without communication, the target can be discovered through random exploration. The success metrics increase over all experiments with increasing communication ranges, up to 1.0 success and 0.925 SPL for the policy trained with $D_S = 2$ and $L = 1$. The policies with $D_S = 2.0$ and $L = 2$ layers perform similar to the policy with $D_S = 4$. The policy with $D_S = \infty$ and $L = 1$ layers performs slightly worse. This can be attributed to the unfiltered inclusion of information from all sensors. Adding locality through a neighborhood and considering multiple neighborhood through multi-hop communication helps in building a more appropriate global representation.

We furthermore test the generalizability to larger environments. The baseline policy has a success rate of 0.619 and an SPL of 0.492, while the fully connected policy has a success rate of up to 0.952 and an SPL of 0.853.

In Fig. 4.9 we further analyze the benefit of communication on our method. We use the policy trained for $D_S = 2$ and $L = 2$ on the small and the large test set and evaluate the performance for a variety of communication ranges and number of sensors. Note that the results in Tab. 4.2 for the large environment are suboptimal due to the constrained communication range, since $D_S = 2.0$ for $L = 2$ covers sensors at most 4.0 m away, while the maximum environment length is 14.0 m. We find that the SPL in the large environment increases approximately linearly from 0.45 for $D_S = 0.0$ to 0.91 for $D_S = 3.5$ and then slightly decreases to a constant of 0.85 for $D_S \geq 4.0$, resulting in a performance increase of $2.0\times$ compared to the communication-free baseline. The small environment performs at 0.72 SPL without communication and 0.93 with communication, resulting in a $1.3\times$ performance increase. Decreasing the number of sensors correspondingly decreases the SPL for a similar minimum and maximum performance. It is to be expected that both environments perform similarly well with the maximum number of sensors and a communication range that results in a coverage of the whole environment, whereas in the no communication case and with only one sensor, the small environment performs better (since less exploration is required to navigate to the target).

Sim-to-Real Policy Transfer

After performing the domain adaptation, we evaluate the performance of the policy on the real-world setup as described in Sec. 4.4.3. We use a motion capture system to create a map

Table 4.2 Evaluation of simulation result for five different policies trained for different communication ranges D_S and number of GNN layers L. We report the success rate and SPL for runs where the target was within line-of-sight (LOS) and outside line-of-sight at the start of the experiment (NLOS) as well as the number of samples M on a small test set as well as a generalization set of a larger environment.

		Training Distribution ($W = 8, H = 10, N = 7$)				Generalization ($W = 16, H = 20, N = 13$)			
		LOS ($M = 198$)		NLOS ($M = 52$)		LOS ($M = 58$)		NLOS ($M = 42$)	
D_S	L	Success	SPL	Success	SPL	Success	SPL	Success	SPL
0.0	1	1.000	0.958	0.827	0.719	1.000	0.945	0.619	0.492
2.0	1	1.000	0.963	1.000	0.925	0.983	0.935	0.833	0.742
4.0	1	1.000	0.963	0.981	0.912	1.000	0.945	0.905	0.804
2.0	2	1.000	0.964	1.000	0.909	1.000	0.954	0.857	0.756
∞	1	1.000	0.962	0.962	0.880	1.000	0.940	0.952	0.853

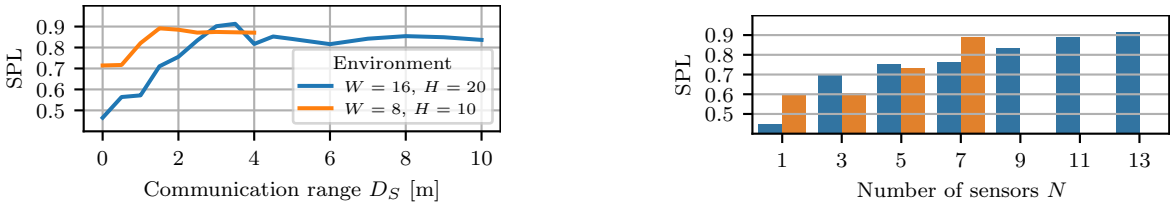


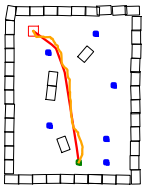
Fig. 4.9 We record the performance of the policy trained for $D_S = 2$ and $L = 2$ for a range of different communication ranges and number of sensors. Left: The large environment (blue) is populated with $N = 13$ sensors and the small environment (orange) with $N = 7$ sensors. Right: The large environment (blue) has a communication range of $D_S = 3.5m$ and the small environment (orange) of $D_S = 1.5m$. Both values result in peak performance for the maximum number of sensors in the corresponding environment, as can be seen on the left side. It can be seen that increasing communication range and number of sensors benefits the SPL.

of the environment and track the robot and target position for the evaluation. We construct three different environments, two of which are taken from the test set (Env. A and B) and a random environment (Env. C) created by us with a similar sensor layout to Env. B but with a different obstacle placement. We run 25 evaluation runs for each environment, resulting in a total of 75 evaluation runs, of which 66 are NLOS runs.

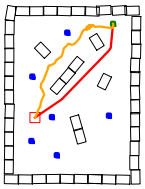
The results for all environments are shown in Fig. 4.10. The total success rate across all environments is 0.745 and 0.577 SPL for NLOS. Environment A is the least challenging environment, with only three small obstacles in total, and has the highest NLOS success rate of 0.895 and SPL of 0.662. Environment B is, with an NLOS success rate of 0.625, the most challenging one, with a total of five obstacles, ranging from small to large, and

Env	LOS Success	LOS SPL	NLOS Success	NLOS SPL
A	0.833	0.679	0.895	0.662
B	0.875	0.791	0.625	0.497
C	1.000	0.897	0.700	0.559
All	0.895	0.783	0.745	0.577

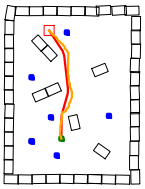
(a) Real-world results



(b) Env. A



(c) Env. B



(d) Env. C

Fig. 4.10 (a) Evaluation of real-world results for three different environments for the fully connected policy. We report success rate and SPL for runs where the target was within line-of-sight (LOS, $M_A = 2$, $M_B = 5$, $M_C = 1$) and outside line-of-sight at the start of the experiment (NLOS, $M_A = 23$, $M_B = 19$, $M_C = 24$). (b, c, d) A selection of two policy evaluations for NLOS configurations for each real-world environment. Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert P and the orange path the path p chosen by the policy π.

narrow passages between obstacles. Environment C is a variation of Environment B and lies in between Environment A and C with an NLOS success rate of 0.7 and an SPL of 0.577. Even though the sim-to-real transfer is generally successful, there is a noticeable reality gap. We found that the real-world sensors have a much smaller visibility range than the simulated sensors, and even though we did not conduct any real-world baseline experiments, it is to be expected that the SPL for $D_S = 0$ (i.e., no communication) would be significantly smaller.

Limitations

All communication between sensors is synchronous, and we do not consider any communication time delays or message dropouts. Asynchronous evaluation and message delays can negatively affect the performance of GNN-based control policies [42]. These effects could potentially be counteracted by incorporating such message delays into the training procedure.

The target must be within line-of-sight of *at least one* sensor within multi-hop communication to the sensor node attached to the mobile robot. In future work, this problem can be resolved by replacing the static sensors with mobile robots that dynamically cover the environment.

Due to limitations of the expert solver, our current approach does not consider dynamic obstacles. This limitation can be alleviated by using a more capable solver or augmenting our learning paradigm to include interactive components (e.g., through Reinforcement Learning).

Lastly, the local coordinates of the robot and all the sensors are aligned, i.e. they are facing the same direction. We do not assume knowledge of the global nor relative positioning of the robot or sensors. This limitation can be circumvented by using a magnetic field sensor

to align the visual sensors or by introducing a random rotation of cameras into the dataset to learn an equivariant mapping of the camera heading. Due to architectural constraints, learning this can be challenging, and rotation equivariant architectures such as [168] have to be considered.

4.5 Discussion and Further Work

In this chapter, we first demonstrated the real-world deployment of a GNN-based policy to a fully decentralized real-world multi-robot system using ROS2 and an Adhoc communication network. We performed a suite of experiments that discuss the selection of suitable networking settings, and subsequently presented results on a real-world scenario requiring tight coordination among robots.

Our results showed that our framework allows for the successful deployment of our control policy in an Adhoc configuration, albeit with a performance that is 22 pp worse in terms of success rate and 9 pp worse in terms of median makespan wrt the centralized mode. Even though the deployment of our scenario was successful, we reported a degradation of performance when moving from simulation to the real world, which can be attributed to real-world effects such as communication delays.

Extending from the ROS2-based system, we propose a vision-only-based learning approach that leverages a Graph Neural Network (GNN) to encode and communicate relevant viewpoint information to the mobile robot. In our experiments, we first demonstrate generalization to previously unseen environments with various sensor network layouts. Our results show that by using communication between the sensors and the robot, we achieve a $1.3\times$ improvement on small environments and a $2.0\times$ improvement in SPL on large environments, when compared to the communication-free baseline, hence showing increasing improvement for larger environment sizes. This is done without requiring a global map, positioning data, nor pre-calibration of the sensor network. The benefit of utilizing communication in wireless sensor networks increases as the size of the environment increases, since the robot is less likely to discover the goal through random (unguided) exploration using the communication-free baseline.

We perform a zero-shot transfer of our model from simulation to the real world. To this end, we train a translator model that translates between real and simulated images so that the navigation policy (which is trained entirely in simulation) can be used directly on the real robot, without additional fine-tuning. Physical experiments demonstrate first-of-a-kind results that show successful real-world demonstrations on a practical robotic platform with raw visual inputs.

In future work, we will evaluate the impact of asynchronous and delayed communication between sensors and the robot. Furthermore, we plan to analyze the impact of small vs large visual overlaps (among sensor nodes) on the overall navigation performance. We will use our software framework to validate novel mechanisms that are robust to communications-specific domain shifts and thus aid in closing the sim-to-real gap for GNNs. We believe that the presented framework will facilitate the deployment of robot systems into more complex environments and the unstructured outdoors, potentially leveraging more complex networking architectures such as mesh networks and on-board sensing.

Chapter 5

Data-driven Heuristic for Path Planning

5.1 Introduction

Path planning is one of the fundamental problems in robotics. It can be formulated as: given a robot and a description of the environment, plan a conflict-free path between the specified start and goal locations. Multi-Agent Path Finding (MAPF) is central to many multi-agent problems. The solution to MAPF is to generate collision-free paths guiding agents from their start positions to designated goal positions.

Conflict-Based Search (CBS) is one of the most popular planners for MAPF [93]. It is provably complete and optimal. However, solving MAPF optimally is NP-hard [169, 170]. Consequently, CBS suffers from scalability, as the search space grows exponentially with the number of agents. Bounded-suboptimal algorithms [14, 171] guarantee a solution that is no larger than a given constant factor over the optimal solution cost. Though these methods often run faster than CBS for grid-based MAPF instances, their effectiveness remains an open question for non-grid-based problem settings, wherein agents can move in an arbitrary continuous domain. In addition, since most of these heuristics rely heavily on collision checking for conflicts, their computational costs may become considerable when the graphs are dense.

Learning-based methods have shown their potential in solving MAPF tasks efficiently [172, 173, 15], which offload the online computational burden into an offline learning procedure. Vanilla GNN-based decentralized path planning [19] has demonstrated its performance empirically via an end-to-end learning approach. However, this black box approaches are arguably deployable in the actual workplace, as they are hard to find a guaranteed and interpretable solution.

To solve this problem, we designed Graph Transformer, as a heuristic function, to accelerate the focal search within Conflict-Based Search (CBS) in a non-grid setting, especially

dense graphs. Such that our framework guarantees both the completeness and bounded suboptimality of the solution.

Re-planning strategies are commonly used to cope with dynamic obstacles, where a planning algorithm searches for an alternative path whenever the robot encounters a conflict. For the explainability and interpretability for RL, we introduced a global path planning algorithm (for example, A*) to generate a globally optimal path, which act as part of the reward function to encourage the robot to explore all potential solutions 'weekly supervised' by the optimal path. This novel reward structure is called globally Guided Reinforcement Learning approach (G2RL), which provides not only the interpretability of our framework but also dense rewards. It does not require the robot to strictly follow global guidance at every step, thus encouraging the robot to explore all potential solutions. As our reward function is independent of the environment, our trained framework can be generalized to arbitrary environments and used to solve the multi-robot path planning problem in a fully distributed reactive manner.

5.2 Background and Related Work

Traditional path planning approaches. Path planning can be divided into two categories: global path planning and local path planning [174]. The former approach includes graph-based approaches (for example, Dijkstra and A* [175]) and sampling-based approaches (for example, RRT and its variant [71]), in which all the environmental information is known to the robot before it moves. For local path planning, at least a part or almost all the information on the environment is unknown. Compared to global path planners, local navigation methods can be very effective in dynamic environments. However, since they are essentially based on the fastest descent optimization, they can easily get trapped in a local minimum [176]. A promising solution is to combine the local planning with global planning, where the local path planner is responsible for amending or optimizing the trajectory proposed by the global planner. For instance, [177] proposed a global dynamic window approach that combines path planning and real-time obstacle avoidance, allowing robots to perform high-velocity, goal-directed, and reactive motion in unknown and dynamic environments. Yet their approach can result in highly sub-optimal paths. The authors in [178] adopt multi-policy decision making to realize autonomous navigation in dynamic social environments. However, in their work, the robot's trajectory was selected from closed-loop behaviors whose utility can be predicted rather than explicitly planned.

Conflict-Based Search (CBS) is one of the most popular planners for MAPF [93]. It is provably complete and optimal. However, solving MAPF optimally is NP-hard [169, 170].

Consequently, CBS suffers from scalability, as the search space grows exponentially with the number of agents. Bounded-suboptimal algorithms [14, 171] guarantee a solution that is no larger than a given constant factor over the optimal solution cost. Though these methods often run faster than CBS for grid-based MAPF instances, their effectiveness remains an open question for non-grid-based problem settings, wherein agents can move in an arbitrary continuous domain. In addition, since most of these heuristics rely heavily on collision checking for conflicts, their computational costs may become considerable when the graphs are dense.

Learning based approaches. Benefiting from recent advances in deep learning techniques, learning-based approaches have been considered as a promising direction to address path planning tasks. Reinforcement Learning (RL) has been implemented to solve the path planning problem successfully, where the robot learns to complete the task by trial-and-error. Traditionally, the robot receives the reward after it reaches the target location [74]. As the environment grows, however, the robot needs to explore more states to receive rewards. Consequently, interactions become more complex, and the learning process becomes more difficult. Other approaches apply Imitation Learning (IL) to provide the robot dense rewards to relieve this issue [15, 179]. However, basing the learning procedure on potentially biased expert data may lead to sub-optimal solutions [180, 19, 66]. Compounding this issue, the robot only receives rewards by strictly following the behavior of expert demonstrations, limiting exploration to other potential solutions. Also, over-fitting remains a problem. This is clearly exemplified in [181], where the robot follows the previously learned path, even when all obstacles have been removed from the environment.

5.3 Accelerating Multi-Agent Planning using Graph Transformers with Near-Optimal Guarantees

In this section, we propose to use a Graph Transformer as a heuristic function to accelerate Conflict-Based Search (CBS) in a non-grid setting. Similar to previous works [173], by introducing focal search to CBS, our framework guarantees both the completeness and bounded-suboptimality of the solution. Our contributions are as follows:

- We propose a novel architecture, i.e., the Graph Transformer, which leverages the underlying structure of the MAPF problem. The proposed architecture has several desired properties, e.g., dealing with an arbitrary number of agents, making it a natural fit for

the MAPF problem. To our knowledge, our work is one of the first works to introduce a learning component to MAPF problems under *non-grid-based* problem settings.

- We design a novel training objective, i.e., Contrastive Loss, to learn a heuristic that ranks the search nodes. Unlike [173], our loss can be directly optimized without introducing an upper bound, which is suitable for deep learning.
- We demonstrate the generalizability of our model by training with relatively few agents and testing in unseen instances with larger agent numbers. Results show that our approach can accelerate CBS significantly while ECBS, using handcrafted heuristics [14], fails.

5.3.1 Problem Formulation

We study Multi-Agent Path Finding (MAPF) in the 2D continuous space $\mathscr{C} \subseteq \mathbb{R}^2$. The configuration space $\mathscr{C}$ consists of a set of obstacles $\mathscr{C}_{obs} \subseteq \mathscr{C}$ and free space $\mathscr{C}_{free} : \mathscr{C} \setminus \mathscr{C}_{obs}$. Note that $\mathscr{C}_{obs}$ could be different from what appears in the workspace, since it also considers the geometric shape of the agent, which may not solely be a point mass.

A random geometric graph $G = \langle V, E \rangle$ is sampled uniformly from the 2D space. Every sampled vertex $v \in V$ is collision-free, i.e., $v \in \mathscr{C}_{free}$. A directed edge $e \in E : (v_i \rightarrow v_j)$ connects v_i to v_j, if (i) v_j is one of the neighbors of v_i, and (ii) the edge is collision-free, i.e., $e \subseteq \mathscr{C}_{free}$. The neighbor set can be defined as the r-radius or k-nearest neighbors.

Suppose there are M agents on this graph G. Each agent i occupies a region $\mathscr{R}(q) \subseteq \mathscr{C}$, associated with a vertex $q \in V$. We assign a start vertex s_i and a goal vertex g_i to each agent i. We denote the path of agent i as $\sigma_i : \{v_i^t\}_{t \in [1 \cdots T_i]}$, where $T_i \in \mathbb{Z}_{>0}$, and agent i is on vertex v_i^t at time step t. We denote e_i^t as the edge $(v_i^t \rightarrow v_i^{t+1})$. In this work, we initially approach the task in a discrete-time domain, aligning with the subsequent definition of time presented in the follow-up content.

Problem Description. We consider a tuple $(G, \mathscr{S}, \mathscr{G}, \mathscr{C}, \mathscr{R})$ as a problem instance of MAPF, where $\mathscr{S} : \{s_i\}_{i \in [1 \cdots M]}$ and $\mathscr{G} : \{g_i\}_{i \in [1 \cdots M]}$ are the start and goal vertices. A conflict-free solution $\{\sigma_i\}_{i \in [1 \cdots M]}$, should satisfy the following objectives, given arbitrary time t and pair of agents i, j [182]:

(Endpoint) $v_i^0 = s_i \wedge v_i^{T_i} = g_i$

(Obstacle) $v_i^t \in \mathscr{C}_{free} \wedge e_i^t \subseteq \mathscr{C}_{free}$

(Inter-agent) $\mathscr{R}(q_i) \cap \mathscr{R}(q_j) = \emptyset$, for all $q_i \in e_i^t, q_j \in e_j^t$

Note: If $t \geq T_i$, we assume the corresponding v_i^t is equal to $v_i^{T_i}$. This means that the agent will stay at the goal starting from time step $v_i^{T_i}$.

Solution Quality. We assume each edge requires one time step to traverse. The quality of the solution is measured by the sum of travel times (flowtime): $\sum_{i \in [1 \cdots M]} T_i$.

5.3.2 Background: Conflict-Based Search with Biased Heuristics

In this section, we first introduce Conflict-Based Search (CBS), an optimal multi-agent planner [93]. Then we introduce *focal search* [183, 184], which incorporates the biased heuristic into the CBS framework, while preserving the guarantees of bounded-suboptimality and completeness [14].

Conflict-Based Search

Conflict-Based Search (CBS) is an optimal bi-level tree search algorithm of MAPF. The high-level planner aims to solve inter-agent conflicts, while the low-level planner aims to generate optimal individual paths. Here we denote an inter-agent conflict as $(i, j, t, v_i^{t-1}, v_j^{t-1}, v_i^t, v_j^t)$, which implies two edges, $(v_i^{t-1} \rightarrow v_i^t)$ and $(v_j^{t-1} \rightarrow v_j^t)$, dissatisfy the inter-agent objective mentioned in Section 5.3.1.

The high-level planner maintains a tree and decides which search node to expand in a best-first manner. To this end, each search node N stores the following information:

(1) A set of constraints $N.\mathscr{T}$. A constraint (i, v, t) indicates that agent i should not traverse to graph vertex v at time t.

(2) A solution $N.\sigma$: $\{\sigma_i\}_{i\in[1\cdots M]}$. The solution satisfies the endpoint and obstacle objective, but it may or may not satisfy the inter-agent objective. In addition, the solution should obey the constraints $N.\mathscr{T}$, i.e., for all$i$, for all$v_i^t \in \sigma_i, (i, v, t) \notin N.\mathscr{T}$.

(3) The cost of the solution $N.c$. The high-level planner prioritizes which search node to expand based on this metric.

On the high level, CBS first creates a root search node with no constraints, then keeps selecting a search node and expanding it. A search node N^* is selected if it is a leaf node with the lowest cost. CBS then checks whether the solution σ of N^* has an inter-agent conflict. If there is no conflict, σ will be returned as the final result. Otherwise, CBS chooses the first conflict C: $(i, j, t, v_i^{t-1}, v_j^{t-1}, v_i^t, v_j^t)$, and splits it into two constraints $C(1)$: (i, t, v_i^t) and $C(2)$: (j, t, v_j^t). Two child search nodes N_1 and N_2 are then generated, with constraints for all$i = 1, 2, N_i.\mathscr{T} = N^*.\mathscr{T} \cup \{C(i)\}$ respectively. Then an optimal low-level planner, e.g., A* [185], is called by each child search node, which replans the path for each affected agent and records the respective solution and cost. CBS guarantees completeness and optimality, since both the high-level and low-level planners are performing best-first search [93].

Incorporating Biased Heuristics using Focal Search

CBS is an optimal planner, but it does not scale well even for grid-based problems settings. To improve the scalability, focal search [183, 184] was introduced by previous works, e.g.,

Bounded CBS (BCBS) and Enhanced CBS (ECBS) [14]. Here we describe a simplified version of BCBS.

We present the pseudocode of focal search in Algorithm 3. Focal search introduces the focal set to the CBS framework. A focal set (*Focal*) maintains a fraction of the leaf search nodes in the CBS tree (i.e., *Open*). We denote *LB* as the lowest solution cost in leaf search nodes. All the leaf search nodes that satisfy a near-optimal solution quality $cost \leq w \cdot LB$ will be added to *Focal*. This new CBS will select a search node from *Focal* to expand, instead of that from *Open*. Compared to the original CBS, the new algorithm also performs the best-first search on *Focal*, but the search priority changes from the solution cost to a new heuristic function ψ. Typically, ψ is a handcrafted function that takes a solution as the input, and outputs a value that prefers solutions with fewer conflicts. We instead use a learned heuristic function based on the **Graph Transformer**.

Algorithm 3: CBS with Biased Heuristics [14, 173]

Input: A MAPF instance and suboptimality factor w
Input: Heuristic function ψ (e.g., Graph Transformer)
Generate the root search node R with an initial solution
Initialize open list $Open \leftarrow \{R\}$
$LB \leftarrow R.cost$, and initialize focal list $Focal \leftarrow \{R\}$
while *Open* is not empty
  $N^* \leftarrow \arg\min_{N \in Focal} \psi(N.solution)$
  $C \leftarrow$ first conflict in $N^*.solution$
  if C does not exist
    return $N^*.solution$
  Remove N^* from *Open* and *Focal*
  if $\min_{N \in Open} N.cost > LB$
    $LB = \min_{N \in Open} N.cost$
    $Focal = \{N \in Open : N.cost \leq w \cdot LB\}$
  Generate two children nodes N_1 and N_2 from node N^*
  Add $C(i)$ to $N_i.constraints$, for $i = 1, 2$
  Call low-level planner to get $N_i.solution$, for $i = 1, 2$
  Add N_i to *Open*, for $i = 1, 2$
  Add N_i to *Focal* if $N_i.cost \leq w \cdot LB$, for $i = 1, 2$
return No solution

Proposition. *Algorithm 3 is complete and bounded-suboptimal with a factor of* $w \geq 1$*, as mentioned in [14].*

Suppose the problem is feasible, but Algorithm 3 does not find a solution given a sufficient time budget. Then for an arbitrary search node N with a feasible solution, there exists an ancestor search node N^p added to *Open* but not expanded. Suppose N^{p*} is the search node

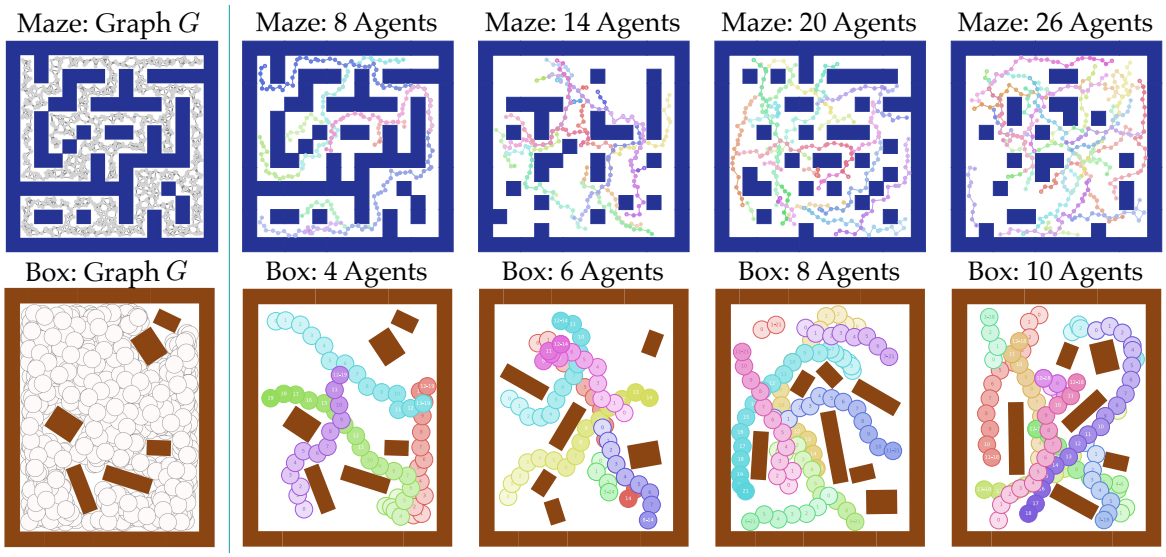


Fig. 5.1 **Left:** Examples of our graph-based MAPF instances. To construct the graph, we sample vertices randomly from the free space and connect them with collision-free edges. **Right:** Problem instances that our approach solves while other baselines fail. Different colors represent the trajectories of different agents. Vertices in the same trajectory have deeper colors if their respective time steps are later.

with the lowest cost among these unexpanded ancestors. N^{p*} is not selected by *Focal*, since it is not expanded. Thus, either (i) N^{p*} is not in *Focal*, or (ii) N^{p*} is in *Focal* but not selected. (i) is impossible, because there do not exist infinitely many solutions that have costs lower than $\frac{1}{w} \cdot N^{p*}.c$. (ii) cannot happen, because there do not exist infinitely many solutions with costs lower than or equal to $w \cdot N^{p*}.c$. As a result, N^{p*} will be selected eventually and expanded. Therefore, we have proved the algorithm to be complete by contradiction. The focal search never expands search nodes with costs higher than w times the optimal solution; therefore, it is bounded-suboptimal with a factor of w.

We note that the focal search described here is a special case of Bounded-CBS [14], i.e., Bounded-CBS $(w, 1)$, as the focal search is only applied to the high-level planner. In our graph-based problem settings, there is no significant improvement when applying the focal search to the low-level planner. Rather, if we introduce the focal search to the low-level planner, it would consume a notable portion of computation on the collision checking of edges, which has no improvement in the overall performance. We refer readers to Question 4 in Section 5.3.4 for further details.

5.3.3 Graph Transformers as Heuristic Functions

In this section, we describe the architecture of the Graph Transformer and how to train it represent a heuristic function that accelerates CBS.

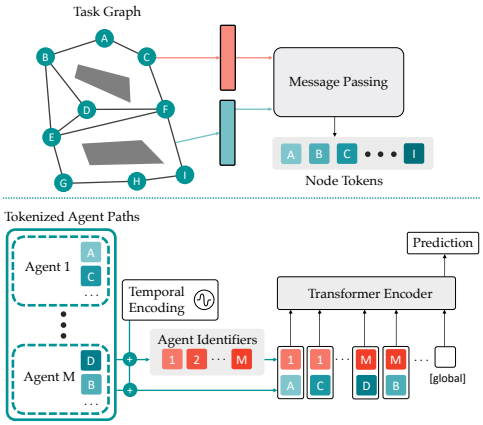


Fig. 5.2 The proposed Graph Transformer architecture. It has several desired properties that are specifically designed to deal with MAPF inputs. See Section 5.3.3 for more details.

Network Architecture

The input to the graph transformer ϕ is a graph G, and a solution $\sigma = \{\sigma_i\}_{i\in[1\cdots M]}$. The output $\phi(G,\sigma)$ predicts a scalar value. The prediction is related to the chance that the current search node will yield descendant search nodes with feasible solutions: If the chance is high, then $\phi(G,\sigma)$ should be low. Otherwise, $\phi(G,\sigma)$ should be high. The graph transformer has two stages: (i) graph tokenization and (ii) attentive aggregation. We describe each stage as follows.

Stage 1: Graph Tokenization. The graph tokenization transforms each graph vertex into an embedding using a Graph Neural Network (GNN). Here we use Message-Passing Neural Networks [105] (MPNN) as the GNN architecture. The input to the MPNN is a graph $G = \langle V,E\rangle$, where the feature d_i^v for each graph vertex $v_i \in V$ is its respective 2D position, and the feature d_l^e for each edge $e_l = (v_i \to v_j)$ is the relative position of v_j to v_i. With two linear layers f_x and f_y, the vertices and edges are first encoded as x and y using for all $v_i \in V, x_i = f_x(d_i^v)$; for all $e_l \in E, y_l = f_y(d_l^e)$. Then, using three MLPs $\{f_k\}_{k\in[1,2,3]}$, the MPNN updates the information for each graph vertex $v_i \in V$ as follows:

$$x_i \leftarrow x_i + \max\{f_k(x_i,x_j,y_l)\}, \text{for all } e_l : (v_i \to v_j) \in E \tag{5.1}$$

After all x_i are updated using f_1, the MPNN continues to update x_i using f_2 and so on. The max denotes the max-pooling over the feature dimension. Since max-pooling can take a set with an arbitrary number of elements and is invariant to the permutation of these elements, the MPNN here can take graphs with an arbitrary number of vertices and edges, but also is permutation invariant by construction.

Stage 2: Attentive Aggregation. After we compute the token x_i for each graph vertex v_i from Stage 1, we model the inter-agent interactions using the Transformer [114]. The path of each agent σ_i is first tokenized as $\rho_i = \{x_j, \text{for all} \, v_j \in \sigma_i\}$. To inject the temporal information of these tokenized solutions, we introduce Temporal Encoding. The approach is similar to [114, 186] (as positional encoding in their settings). We denote $\rho_i^t \in \mathbb{R}^D$ as the vertex token of agent i at time step t. For each token ρ_i^t, we add it element-wisely with a temporal encoding $\rho_i^t \leftarrow \rho_i^t + TE(t) \in \mathbb{R}^D$. The $2k$-th and $2k$+1-th dimensions of $TE(t)$ are as follows:

$$TE(t)_{2k} = \sin(t/10000^{2k/D}) \tag{5.2}$$

$$TE(t)_{2k+1} = \cos(t/10000^{2k/D}) \tag{5.3}$$

The hyperparameter 10000 is used following the common practice [114]. To encourage the model to be aware of which agent each token ρ_i^t belongs to, we concatenate the Agent Identifier τ_i to each token $\rho_i^t \leftarrow \rho_i^t || \tau_i$, similar to [187]. For each agent i, the agent identifier τ_i is calculated by taking the max-pooling over all its vertex tokens: $\tau_i = \max\{\rho_i^t\}, \text{for all} t \in [1 \cdots T_i]$. Then, all tokens from the solution of all agents $\{\rho_i^t : \text{for all} i \in [1 \cdots M], \text{for all} t \in [1 \cdots T_i]\}$ will be fed as the input to the Transformer Encoder. For global prediction, we append an extra trainable token *global* to the input, following the common practice [188, 189]. The Transformer Encoder predicts an output for each input token, and we use the output of the *global* token as ζ. With a linear layer $f_\phi : \mathbb{R}^D \to \mathbb{R}$, the final output is computed as $\phi(G, \sigma) = f_\phi(\zeta)$.

Here we use the Transformer Encoder, since it enables the Graph Transformer to take a variable number of tokens and model their dependencies, while preserving the invariance to the permutation of tokens. For more details on the Transformer Encoder, we refer readers to [114].

Properties of the Graph Transformer. By construction, the Graph Transformer is able to handle the input graph with a variable number of vertices and edges, agents with a variable total number, and the input solution with a variable length. Additionally, it is aware of the temporal information, and inter-agent interactions. Its output is permutation invariant to both the orders of graph vertices and the agents.

Training Graph Transformers

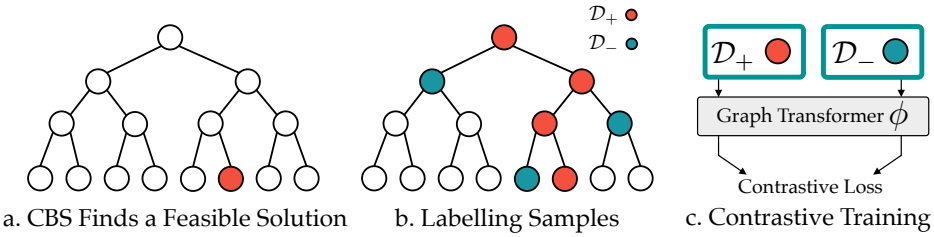


Fig. 5.3 The training framework. We use a supervised Contrastive Loss. The labels are generated from the CBS search tree.

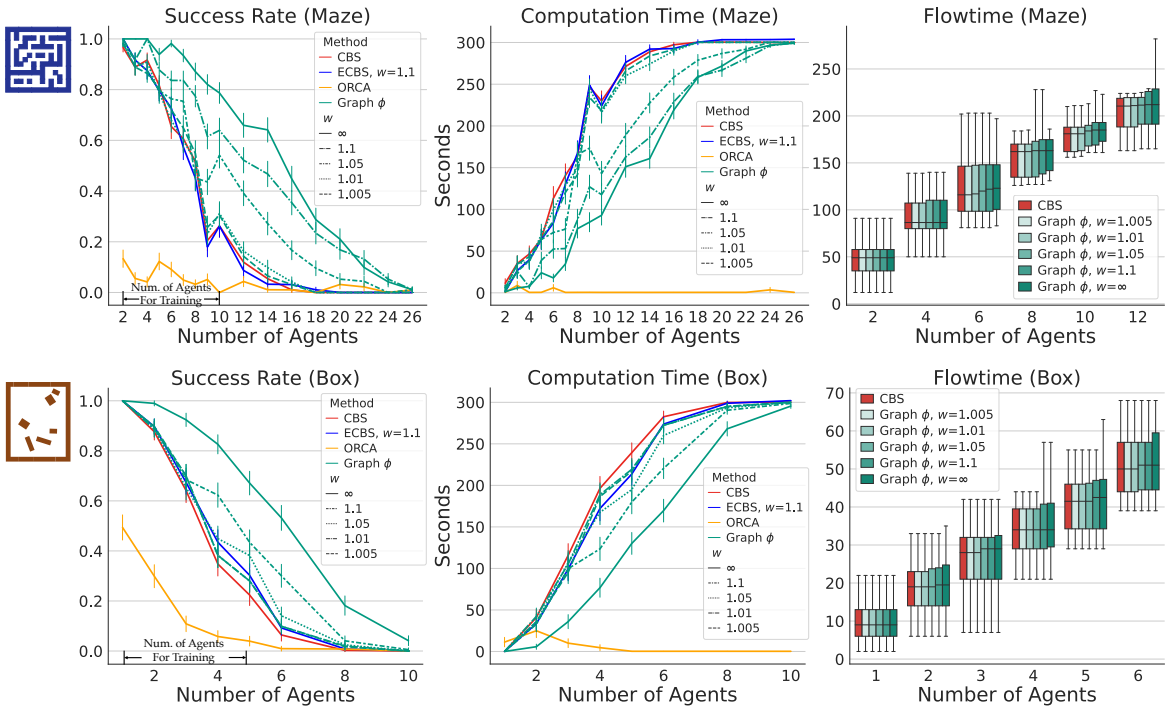


Fig. 5.4 Success rates, computation time, and flowtime within the runtime limit of 5 minutes, as functions of the number of agents. The results are averaged over 100 test instances for each setting of the agent number. We evaluate our approach with $w \in [1.005, 1.01, 1.05, 1.1, \infty]$, and compare its performance with CBS, ECBS ($w = 1.1$) and ORCA. Though trained with relatively few agents, results have shown that our approach generalizes well and significantly outperforms the baselines (CBS, ECBS, and ORCA).

Data Generation. Given a MAPF instance, we first use CBS to generate feasible solutions. No data will be collected if CBS fails to solve the instance. If it succeeds, we start to collect positive and negative samples from its search tree. A positive sample will be collected by

dataset $\mathscr{D}_+$, if itself or one of its descendant search nodes contains the feasible solution. A negative sample will be collected by dataset $\mathscr{D}_-$, if it is a sibling search node of a positive sample. We record each sample's graph G and solution σ. In addition, we record the value d as its respective depth in the search tree.

Supervised Contrastive Learning. The objective of the model is to learn a ranking of the samples. Namely, given an arbitrary pair of positive sample (G, σ_+, d_+) and a negative sample (G, σ_-, d_-) from the same MAPF graph G, we learn the following ranking:

$$\phi(G, \sigma_+) < \phi(G, \sigma_-), \text{ if } d_+ \geq d_-$$

We illustrate the intuition here. Imagine such ranking is learned perfectly and $w = \infty$, meaning all the leaf search nodes will be in *Focal*. Suppose at some time point, *Focal* includes one positive sample p_+. *Focal* may or may not include negative samples. If they exist, then their depths are no deeper than $p_+.d$. Recalled that d is its respective depth in the search tree. Define this condition as a loop invariant. Then p_+ will be selected and expanded first according to the ranking. If p_+ is the feasible solution, then the algorithm terminates. Otherwise, *Focal* will have one positive sample p'_+ and some negative samples, and all negative samples have depths no deeper than $p'_+.d$. Thus, the loop invariant remains true. The loop invariant is also true for the base case, where *Focal* only has the root search node. Therefore, by learning such ranking, we encourage Algorithm 3 to expand positive samples first and expand the negative samples as few as possible, which could save significant computation and greatly accelerate CBS.

We use supervised contrastive learning to train a Graph Transformer ϕ that ranks the positive samples above the negative samples. Given an arbitrary pair of positive and negative samples, $p_+ : (G_+, \sigma_+, d_+) \in \mathscr{D}_+, p_- : (G_-, \sigma_-, d_-) \in \mathscr{D}_-$, this pair is defined to be valid as: $\mathbb{I}(p_+, p_-) : (G_+ = G_-) \wedge (d_+ \geq d_-)$. With a hyperparameter $\gamma = 0.1$, we define $\delta(x) : \max(0, \gamma + x)$. We aim to minimize the Contrastive Loss as follows:

$$\frac{1}{L} \sum_{\substack{p_+ \in \mathscr{D}_+ \\ p_- \in \mathscr{D}_-}} \delta(\phi(G_+, \sigma_+) - \phi(G_-, \sigma_-)) \cdot \mathbb{I}(p_+, p_-) \tag{5.4}$$

where $L = \sum_{p_- \in \mathscr{D}_-}^{p_+ \in \mathscr{D}_+} \mathbb{I}(p_+, p_-)$ is the total number of valid sample pairs.

Once the training of Graph Transformer ϕ reaches convergence, we could deploy it as the heuristic function ψ in Algorithm 3. However, in practice, we found that the model would predict relatively low values for multiple search nodes in the focal set, indicating that all of them may lead to feasible solutions. To break the tie, we instead represent ψ as the depth d combined with ϕ, i.e., $\psi = \langle -d, \phi \rangle$. It means that the algorithm would first prefer the search

nodes with deeper depths; if there exist multiple search nodes with the deepest depths, then it would prefer the search nodes with lower ϕ. Such a design enables the algorithm to make decisions consistently if multiple promising nodes exist. Without further specification, we denote our approach with such a heuristic as **Graph ϕ**.

5.3.4 Experiments

Main Experiments

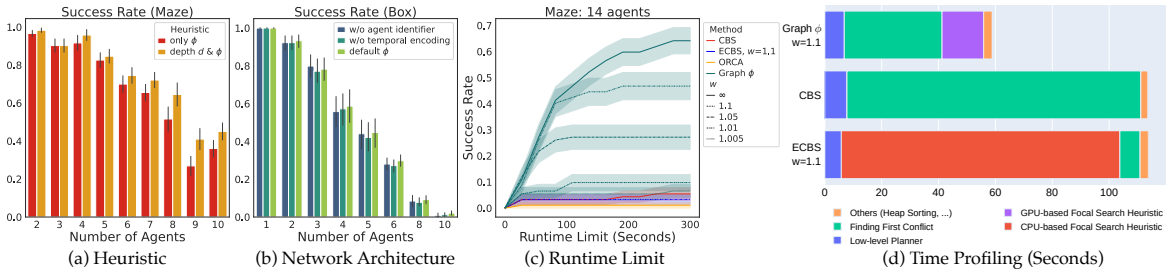


Fig. 5.5 We conduct 4 various ablation studies to evaluate the proposed method systematically. See Section 5.3.4 for more details.

Experimental Setup. We design two types of environments for evaluation: Maze and Box (see Fig. 5.1). Each environment includes 2700 MAPF instances with samples generated by CBS for training. For testing, there are 100 MAPF test instances w.r.t. each agent number setting. We generate a random map for each Maze instance and a set of random obstacles for each Box instance. The graph and start and goal vertices are also generated randomly for each instance.

The training instances vary between 2 and 10 agents for Maze, and vary between 1 and 5 agents for Box. The test instances vary between 2 and 26 agents (2-10, 12, 14, 16, 18, 20, 22, 24, 26) for Maze, and vary between 1 and 10 agents (1-5, 6, 8, 10) for Box. We ensure that the test instances are unseen in the training set. All experiments were conducted using a 12-core, 3.2Ghz i7-8700 CPU and 4 Nvidia GTX 1080Ti GPUs. We test $w \in [1.005, 1.01, 1.05, 1.1, \infty]$ for our method. For all methods, we set the runtime limit as 5 minutes, following the common practice [173].

Baselines. We compare our method with 3 baselines: (i) Conflict-Based Search. (ii) Enhanced Conflict-Based Search (ECBS): a bounded-suboptimal version of CBS [14]. It applies focal search to both high-level and low-level planners, using hand-crafted heuristic functions. We choose its $w = 1.1$. (iii) ORCA [1]: a reactive collision avoidance algorithm, which works effectively in low density environments.

We find that there are very few open-source implementations that could directly apply CBS and ECBS to non-grid-based problems. As a result, we implement all tree-search methods (CBS, ECBS, and our method) from scratch using Python. We use basically the same framework when implementing CBS, ECBS, and our approach. To make the comparison fair, we ensure that all these 3 methods are aggressively optimized by strictly following the C++ implementations [1] and original paper [93, 14], and using Bayesian hyperparameter search for the w of ECBS.

Evaluation Metrics. Our evaluation includes 3 metrics: 1) *Success Rate*, the ratio of the number of successful instances to the total number of test instances. An instance is successful if all agents reach their goals with no collision before the timeout happens. 2) *Computation Time*, the average computational time for each instance, including the failed ones. [2] 3) *Flowtime*, which only compares the solution quality of the optimal CBS to our method, given the instances where both methods succeed, to validate our approach's bounded-suboptimality guarantees.

Overall Performance. We demonstrate the overall performance in Figure 5.4. Our method significantly outperforms the three baselines in both Maze and Box. It requires much less computation time and achieves significantly higher success rates. Furthermore, though only trained with relatively few numbers of agents, our method generalizes remarkably well to a higher number of agents. For example, our network trained by 2 to 10 agents can be generalized up to 26 agents in Maze, while our policy trained by 1 to 5 agents can be generalized up to 10 agents in Box. In particular, with $w = 1.1$, our method achieves the success rates of $47\%, 23\%, 10\%, 1\%$ in Maze with 14, 18, 22, and 26 agents, and achieves the success rates of $62\%, 30\%.4\%, 0.5\%$ in Box with 4, 6, 8, and 10 agents. On the other side, CBS fails to solve Maze with 18 agents and Box with 10 agents within the timeout. Similarly, ECBS starts to fail in all instances with 20 agents for Maze and with 10 agents for Box. In addition, since the bounded-suboptimality of our algorithm is proved theoretically, it is not surprising to see that the solution qualities (flowtime) of our method are very close to the optimal solutions. Finally, ORCA easily fails in highly dense environments, and such performance is consistent with previous works [190].

Ablation Study

In this section, we investigate four questions:

[1] https://github.com/whoenig/libMultiRobotPlanning/

[2] Since CBS and ECBS call different low-level planners (A* and A*-ϵ), we consider the computation time to be the fairest metric to evaluate the efficiency, instead of counting the number of expanded nodes, for instance.

(a) Is incorporating depth information into the heuristic beneficial to the performance? Fig. 5.5 (a) illustrates the comparison between the heuristic with only ϕ, and with depth d and ϕ, on tests from 2 to 10 agents in Maze. The result shows the effectiveness of introducing depth information. The improvement becomes more noteworthy as the number of agents grows, which validates our choice.

(b) Do the Temporal Encoding and the Agent Identifier improve the performance? Fig. 5.5 (b) demonstrates the performances in Box with and without the Temporal Encoding and Agent Identifier. The results show that Agent Identifier and Temporal Encoding improve the success rate. The average improvement of introducing Agent Identifier and Temporal Encoding over the non-default settings is $16\pm33\%$.

(c) How will the runtime limit affect the performance? We take the tests with 14 agents in Maze as an example. We set different runtime limits from 25 seconds to 300 seconds with the interval as 25 seconds. In Fig. 5.5 (c), we show that our methods outperform the baselines regardless of how the runtime limit changes. When the limit is 300 seconds, our method achieves a success rate of 64% ($w = \infty$), while CBS, ECBS, and ORCA only have 5%, 3%, and 1% respectively.

(d) Time profiling each module of the planners. Finally, we wish to answer the question of why the performances of ECBS considerably degrade once we apply it to non-grid-based MAPF instances, compared to the traditional grid-based settings. We profile each method, and average the results over the Maze tests with 2-10 agents.

Fig. 5.5 (d) illustrates that ECBS spends considerable computation on the focal heuristic calculation. Such behavior is reasonable, since now for dense graphs, the heuristic is calculated by checking the collisions along edges. Such collision checking is often the main bottleneck for planning and by itself NP-hard in general [191, 192]. Meanwhile, our method uses a learned function for the focal heuristic calculation (the GPU part), which only takes 15% computation cost compared with ECBS. Compared with CBS and ECBS, our method only requires 50% of the total computation time.

5.4 Mobile Robot Path Planning in Dynamic Environments through Globally Guided Reinforcement Learning

In order to overcome the limitation of sparse reward in larger environment, we develop a hierarchical path-planning algorithm that combines a *global guidance* and a *local RL-based planner*. Concretely, we first utilize a global path planning algorithm (for example, A*) to obtain a globally optimal path, which we refer to as the *global guidance*. During robot motion,

the *local RL-based planner* generates robot actions by exploiting surrounding environmental information to avoid conflicts with static and dynamic obstacles, while simultaneously attempting to follow the fixed global guidance.

Our main contributions include:

- We present a hierarchical framework that combines global guidance and local RL-based planning to enable end-to-end learning in dynamic environments. The local RL planner exploits both spatial and temporal information within a local area (e.g., a field of view) to avoid potential collisions and unnecessary detours. Introducing global guidance allows the robot to learn to navigate towards its destination through a *fixed-sized* learning model, even in large-scale environments, thus ensuring scalability of the approach.
- We present a novel reward structure that provides dense rewards, while not requiring the robot to strictly follow the global guidance at every step, thus encouraging the robot to explore all potential solutions. In addition, our reward function is independent of the environment, thus enabling scalability as well as generalizability across environments.
- We provide an application of our approach to multi-robot path planning, whereby robot control is fully distributed and can be scaled to an arbitrary number of robots.
- Experimental results show that our single-robot path planning approach outperforms local and global re-planning methods, and that it maintains consistent performance across numerous scenarios, which vary in map types and number of obstacles. In particular, we show that our application to multi-robot path planning outperforms current state-of-the-art distributed planners. Notably, the performance of our approach is shown to be comparable to that of centralized approaches, which, in contrast to our approach, assume global knowledge (i.e., trajectories of all dynamic objects).

5.4.1 Problem Description

Environment representation. Consider a 2-dimensional discrete environment $\mathscr{W} \subseteq \mathbb{R}^2$ with size $H \times W$ and a set of N_s static obstacles $\mathscr{C}_s = \{s_1, ..., s_{N_s}\}$, where $s_i \subset \mathscr{W}$ denotes the i^{th} static obstacle. The free space $\mathscr{W} \setminus \mathscr{C}_s$ is represented by a roadmap $G = \langle \mathscr{C}_f, \mathscr{E} \rangle$, where $\mathscr{C}_f = \{c_1, ..., c_{N_f}\} = \mathscr{W} \setminus \mathscr{C}_s$ represents the set of free cells and $e_{ij} = (c_i, c_j) \subset \mathscr{E}$ represents the traversable space between free cells c_i and c_j that does not cross any other cell (the minimum road segment). In this work, we initially approach the task in a discrete-time domain, aligning with the subsequent definition of time presented in the follow-up content. The set of dynamic obstacles $\mathscr{C}_d(t) = \{d_1(t), ..., d_{N_d}(t)\}$ denotes the position of N_d dynamic obstacles at time t, where for alli, j, t, $d_i(t) \subset \mathscr{C}_f$, $(d_i(t), d_i(t+1)) \subset \mathscr{E}$ or $d_i(t) = d_i(t+1)$,

and $d_i(t) \neq d_j(t)$. In addition, if $d_i(t+1) = d_j(t)$, then $d_j(t+1) \neq d_i(t)$, i.e., any motion conflict should be avoided.

Global guidance. A *traversable path* $\mathscr{G} = \{g(t)\}$ is defined by the following rules: 1) the initial location $g(0) = c_{start}$ and there is a time step t_f that for all$t \geq t_f$, $g(t) = c_{goal}$. Note that, $c_{start}, c_{goal} \in \mathscr{C}_f$; 2) for all$t$, $g(t) \in \mathscr{C}_f$, $(g(t), g(t+1)) \in \mathscr{E}$. The *global guidance* $\mathscr{G}^*$ is the shortest traversable path, defined as $\mathscr{G}^* = \arg\min_{t_f} \mathscr{G}$, between the initial location c_{start} and the goal c_{goal}. Note that the global guidance is generated by A^* here and it can be other shortest path algorithms. It may not be unique in the discrete world, and therefore, we randomly choose one instance in this work.

Assumptions. We assume that the robot knows the information of all the static obstacles and calculates the global guidance at the start of each run. Note that the global guidance is only calculated once and remains the same. During robot motion, we assume that the robot can obtain its global location in the environment and acquire global guidance information. However, we do not assume that the trajectories of dynamic obstacles are known to the robot. The robot can only obtain the current location of dynamic obstacles when they are within its local field of view.

Local observation. The robot has a local field of view (FOV) within which it observes the environment. More specifically, at each time step t, the robot collects the local observation $O_t = \{o_t^f, o_t^s, o_t^d\}$ which is a collection of the location of free cells o_t^f, static obstacles o_t^s and dynamic obstacles o_t^d, within the local FOV with the size of $H_l \times W_l$. In addition, we also define a local segment of the global guidance $\mathscr{G}^*$ as the local path segment $\mathscr{G}_t^*$, which is located within the robot local FOV. Both O_t and $\mathscr{G}_t^*$ are considered as the system input information at each time t.

Robot action. The robot action set is defined as $\mathscr{A} = \{\text{Up}, \text{Down}, \text{Left}, \text{Right}, \text{Idle}\}$, i.e., at each time step, the robot can only move to its neighboring locations or remain in its current location.

Objective. Given as input the local segment $\mathscr{G}_t^*$ of the global guidance, the current local observation $\mathscr{O}_t$, and a history of local observations $\mathscr{O}_{t-1}$,..., $\mathscr{O}_{t-(N_t-1)}$, output an action $a_t \subset \mathscr{A}$ at each time step t that enables the robot to move from the start cell c_{start} to the goal cell c_{goal} with the minimum number of steps while avoiding conflicts with static and dynamic obstacles.

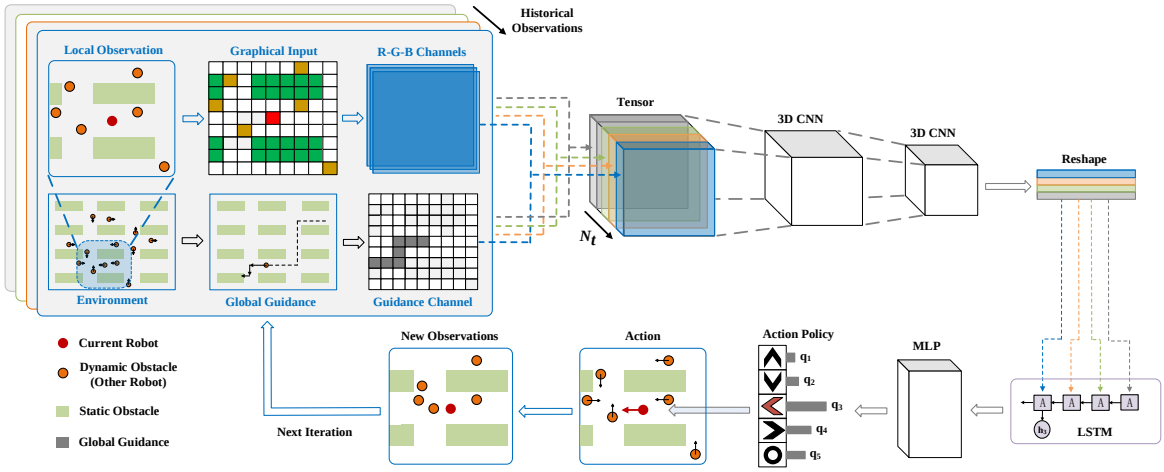


Fig. 5.6 The overall structure of our method. The input of each step is the concatenation of the transformed local observation and the global guidance information. A sequence of historical inputs is combined to build the input tensor of the deep neural network, which outputs a proper action for the robot.

5.4.2 RL-Enhanced Hierarchical Path Planning

In this section, we first describe the overall system structure and then present details of our approach.

System Structure

Figure 5.6 illustrates the overall system structure, which shows that our local RL planner contains four main modules:

1. *Composition of network input:* Firstly, we transform the local observation O_t into a graphic, and use its three channels (RGB data) in combination with a guidance channel to compose the current input. Then a sequence of historical inputs is used to build the final input tensor;

2. *Spatial processing:* Secondly, we utilize a series of 3D CNN layers to extract features from the input tensor, then reshape the feature into N_t one-dimensional vectors;

3. *Temporal processing:* Thirdly, we use an LSTM layer to further extract the temporal information by aggregating the N_t vectors;

4. *Action policy:* Finally, we use two fully connected (FC) layers to estimate the quality q_i of each state-action pair and choose the action $a_i \subset \mathscr{A}$ with $\max_i q_i$.

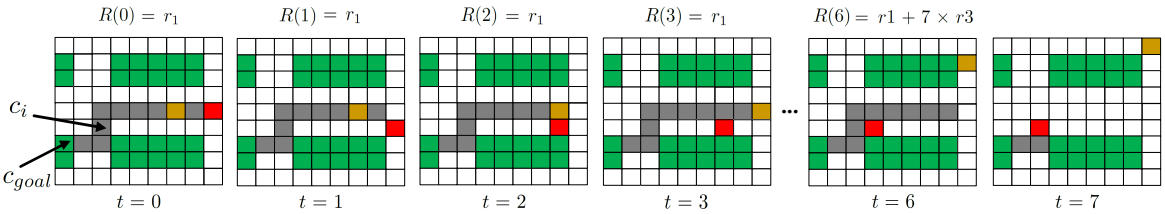


Fig. 5.7 An illustration of our reward function. The green, black, yellow and red cells represent the static obstacle, the global guidance, the dynamic obstacle and the robot, respectively. At $t = 7$, the robot reaches a global guidance cell c_i and receives the reward based on the number of eliminated guidance cells.

Global Guidance and Reward Function

The main role of global guidance is to provide long-term global information. By introducing this information, the robot receives frequent feedback signals in arbitrary scenarios, no matter how large the environment and how complex the local environments are. We achieve this by proposing a novel reward function that provides dense rewards while simultaneously not requiring the robot to follow global guidance strictly. In this manner, we encourage the robot to explore all the potential solutions while also promoting the convergence of learning-based navigation.

More specifically, at each step, the reward function offers: 1) A small negative reward $r_1 < 0$ when the robot reaches a free cell which is not located on the global guidance; 2) A large negative reward $r_1 + r_2$ when the robot conflicts with a static obstacle or a dynamic obstacle, where $r_2 < r_1 < 0$; 3) A large positive reward $r_1 + N_e \times r_3$ when the robot reaches one of the cells on the global guidance path, where $r_3 > |r_1| > 0$ and N_e is the number of cells removed from the global guidance path, between the point where the robot first left that path, to the point where it rejoins it.

The reward function can be defined formally as:

$$R(t) = \begin{cases} r_1 & \text{if } c_r(t+1) \in \mathscr{C}_f \setminus \mathscr{G}^* \\ r_1 + r_2 & \text{if } c_r(t+1) \in \mathscr{C}_s \cup \mathscr{C}_d(t+1) \\ r_1 + N_e \times r_3 & \text{if } c_r(t+1) \in \mathscr{G}^* \setminus \mathscr{C}_d(t+1) \end{cases} \tag{5.5}$$

where $c_r(t+1)$ is the robot location after the robot takes the action a_t at time t, $R(t)$ is the reward value of action a_t.

Figure 5.7 shows an example of our reward function. At $t = 0$, since there is a dynamic obstacle in front, our RL planner moves the robot to the lower cell to avoid conflict. From $t = 1$ to $t = 6$, the robot does not need to return to the global guidance path immediately, but can continue to move left until it reaches one cell c_i located on the global guidance path at

$t = 7$. Here $R(0) = R(1) = R(2) = R(3) = R(4) = R(5) = r_1$ and $R(6) = r_1 + 7 \times r_3$, since at $t = 7$, $N_e = 7$ cells have been removed in the global guidance.

We remove the path segment up to c_i as soon as the robot obtains the reward $R(6)$ to ensure that the reward of each cell on the global guidance path can only be collected once. The removed guidance cells will be marked as normal free cells (white cells) in the graphic images inputted in the following iterations. In contrast to IL-based methods, we do not require the robot to strictly follow the global guidance at each step, since the robot receives the same cumulative reward as long as it reaches the same guidance cell given from the same start cell. As a result, our model can also be trained from scratch without IL, which circumvents potentially biased solutions [180].

In the training process, we stop the current episode once the robot deviates from the global guidance too much, namely, if no global guidance cell can be found in the FOV of the robot.

Local RL Planner

As shown in Figure 5.6, we transform a robot's local observation into an *observation image* defined as: 1) The center pixel of the image corresponds to the current robot position, the image size is the same as the robot's FOV $H_l \times W_l$, i.e., one pixel in the image corresponds to one cell in the local environment; 2) All the static and dynamic obstacles observed are marked in the image, where we use one color to represent static obstacles and another color to denote dynamic ones. In addition, a guidance channel that contains the local path segment $\mathscr{G}_t^*$ of the global guidance is introduced to combine with the three channels (RGB data) of the observation image, to compose the current input $\mathscr{I}_t$. Our RL planner takes the sequence $\mathscr{S}_t = \{\mathscr{I}_t, \mathscr{I}_{t-1} ... \mathscr{I}_{t-(N_t-1)}\}$ as the inputs at step t.

We use Double Deep Q-Learning (DDQN) [193] for our RL planner. At time step t, the target value $Y_t = R(t)$, if $c_{t+1} = c_{goal}$, otherwise,

$$Y_t = R(t) + \gamma Q(\mathscr{S}_{t+1}, \arg\max_{a_t} Q(\mathscr{S}_{t+1}, a_t; \theta); \theta^-) \tag{5.6}$$

where $Q(.)$ is the quality function, θ and θ^- are the current and target network parameters respectively, γ is the discount value and $R(t)$ is defined in (5.5). To update the parameters θ, we sample N_b transitions from the replay buffer, and define the loss $\mathscr{L}$ as:

$$\mathscr{L} = \frac{1}{N_b} \sum_{j=1}^{N_b} (Y_t^j - Q(\mathscr{S}_t^j, a_t^j; \theta))^2. \tag{5.7}$$

As shown in Figure 5.6, our model is comprised of 3D CNN layers followed by LSTM and FC layers. The input is a five-dimensional tensor with size $N_t \times H_l \times W_l \times 4$, where N_b is the batch size sampled from the replay buffer.

We first use 3D CNN layers to extract spatial information. The size of the convolutional kernel is $1 \times 3 \times 3$. We stack one CNN layer with kernel stride $1 \times 1 \times 1$ and another with stride $1 \times 2 \times 2$ as a convolutional block, which repeats N_c times for downsampling in the spatial dimension. The embeddings extracted by the CNN layers are reshaped into N_t one-dimensional vectors and fed into the LSTM layer to extract temporal information. Two FC layers are attached to the LSTM layer, where the first layer (followed by a `ReLU` activation function) reasons about the extracted information and the second layer directly outputs the value q_i of each state-action pair $(\mathscr{S}_t, a_i)$ with a `Linear` activation function.

Application to Reactive Multi-Robot Path Planning

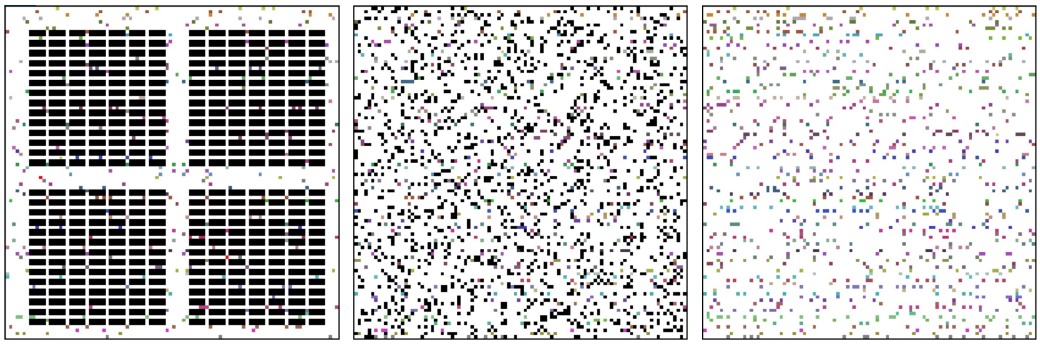
The main idea of our local RL planner is to encourage the robot to learn how to avoid surrounding obstacles while following global guidance. Since robots only consider local observations and global guidance, and do not need to explicitly know any trajectory information nor motion intentions of other dynamic obstacles or robots, the resulting policy can be copied onto any number of robots and, hence, scales to arbitrary numbers of robots. Based on the aforementioned rationale, our approach is easily extended to resolving the multi-robot path planning problem in a fully distributed manner, whereby dynamic obstacles are modeled as independent mobile robots. Each robot considers its own global guidance and local observations to generate actions. Since we do not require communication among robots, this is equivalent to an uncoordinated reactive approach. Note that the above extension is fully distributed, can be trained for a single agent (i.e., robot) and directly used by any number of other agents.

5.4.3 Implementation

In this section, we introduce the network parameters and describe our training and testing strategies.

Model Parameters

In the experiments, we use the A* algorithm to generate the global guidance. The default parameters are set as follows: robot local FOV size $H_l = W_l = 15$, the length of input sequence $N_t = 4$, the reward parameters $r_1 = -0.01$, $r_2 = -0.1$, and $r_3 = 0.1$. The convolutional block is repeated $N_c = 3$ times with input batch size $N_b = 32$. The activation functions are `ReLU`.



(a) Regular map (b) Random map (c) Free map

Fig. 5.8 Map examples. The black and colored nodes represent the static and dynamic obstacles respectively. Map parameters can be found in Section 5.4.3.

We use 32 convolution kernels in the first convolutional block and double the number of kernels after each block. After the CNN layers, the shape of feature maps is $4 \times 2 \times 2 \times 128$. In the LSTM layer and the two FC layers, we use 512, 512, and 5 units, respectively.

Environments

As shown in Figure 5.8, we consider three different environment maps to validate our approach, i.e., a regular map, a random map, and a free map. The first one imitates warehouse environments and contains both static and dynamic obstacles, where the static obstacles are regularly arranged and the dynamic ones move within the aisles. In the random map, we randomly set up a certain density of static obstacles and dynamic obstacles. In the free map, we only consider a certain density of dynamic obstacles. The default size of all the maps is 100×100. The static obstacle density in each map is set to 0.392, 0.15, and 0, respectively, and the dynamic obstacle density is set to 0.03, 0.05, and 0.1, respectively.

Dynamic obstacles are modeled as un-controllable robots that are able to move one cell at each step in any direction. Their start/goal cells are randomly generated, and their desired trajectories are calculated through A* by considering the current position of any other obstacles. During training and testing, each dynamic obstacle continuously moves along its trajectory, and when motion conflict occurs, it will: 1) with a probability of 0.9, stay in its current cell until the next cell is available; 2) otherwise, reverse its direction and move back to its start cell.

Training and Testing

We train our model with one NVIDIA GTX 1080ti GPU in Python 3.6 with TensorFlow 1.4 [194]. The learning rate is fixed to 3×10^{-5}, and the training optimizer is `RMSprop`. We use ε-greedy to balance exploration and exploitation. The initial value is set to be $\varepsilon = 1$ and decreases to 0.1 linearly when the total training steps reach $200,000$. In training, we randomly choose one of the three maps as the training map and configure the dynamic obstacles by following the settings in Section 5.4.3. Then we randomly select two free cells as the start and goal cells. During the training, we end the current episode and start a new one if one of the following conditions is satisfied: 1) the number of training steps in the current episode reaches a maximum defined as $N_m = 50 + 10 \times N_e$; 2) the robot can not obtain any global guidance information in its local FOV; 3) the robot reaches its goal cell c_{goal}. In addition, after the robot completes 50 episodes, the start and goal cells of all the dynamic obstacles are re-randomized.

Before learning starts, one robot explores the environment to fill up the replay buffer, which is comprised of a `Sumtree` structure to perform prioritized experience replay [195]. Note that `Sumtree` is a binary tree, which computes the sum of the values of its children as the value of a parent node. In each episode, the robot samples a batch of transitions from the replay buffer with prioritized experience replay (PER) based on the calculated temporal difference error by the DDQN algorithm. During testing, all methods are executed on an Intel i7-8750H CPU.

Performance Metrics

The following metrics are used for performance evaluation:

- *Moving Cost*:

$$\text{Moving Cost} = \frac{N_s}{||c_{goal} - c_{start}||_{L1}} \tag{5.8}$$

  where N_s is the number of steps taken and $||c_{goal} - c_{start}||_{L1}$ is the Manhattan distance between the start cell and the goal cell. This metric is used to indicate the ratio of actual moving steps w.r.t the ideal number of moving steps without considering any obstacles.

- *Detour Percentage*:

$$\text{Detour Percentage} = \frac{N_s - L_{A^*}(c_{start}, c_{goal})}{L_{A^*}(c_{start}, c_{goal})} \times 100\% \tag{5.9}$$

Table 5.1 Single robot path planning results: Moving costs and detour percentages of different approaches. Values are listed as “mean (standard deviation)" across 100 instances. The lowest (best) values are highlighted.

	Moving Cost				Detour Percentage				Computing Time (s)		
	Local	Global	Naive	Ours	Local	Global	Naive	Ours	Local	Global	Ours
Regular-50	1.58(0.50)	1.31(0.29)	1.38(0.35)	**1.18(0.16)**	36.7(46)%	23.7(27)%	31.5(34)%	**15.2(15)**%	0.004	**0.003**	0.011
Regular-100	1.57(0.35)	1.23(0.21)	1.42(0.31)	**1.12(0.12)**	36.3(34)%	18.7(20)%	39.5(30)%	**10.7(12)**%	0.005	**0.004**	0.012
Regular-150	1.50(0.32)	1.19(0.14)	1.36(0.26)	**1.09(0.08)**	33.3(32)%	16.0(14)%	35.0(36)%	**8.2(8)**%	0.007	**0.004**	0.015
Random-50	1.35(0.28)	1.28(0.18)	1.36(0.28)	**1.21(0.13)**	25.1(26)%	21.1(17)%	30.1(35)%	**16.7(12)**%	0.005	**0.004**	0.013
Random-100	1.43(0.27)	1.26(0.14)	1.34(0.29)	**1.15(0.10)**	30.0(26)%	20.5(14)%	32.3(34)%	**13.0(10)**%	0.006	**0.006**	0.015
Random-150	1.37(0.17)	1.17(0.09)	1.40(0.28)	**1.11(0.08)**	27.0(17)%	14.5(9)%	39.4(40)%	**9.1(8)**%	0.010	**0.006**	0.018
Free-50	1.27(0.20)	1.24(0.15)	1.31(0.24)	**1.14(0.09)**	21.2(20)%	19.4(15)%	28.9(23)%	**12.3(9)**%	0.008	**0.007**	0.018
Free-100	1.31(0.13)	1.21(0.11)	1.34(0.25)	**1.11(0.07)**	23.6(13)%	17.3(11)%	33.6(25)%	**9.1(7)**%	0.015	**0.011**	0.022
Free-150	1.27(0.12)	1.14(0.06)	1.32(0.22)	**1.07(0.05)**	21.2(12)%	12.3(6)%	31.5(22)%	**6.5(5)**%	0.017	**0.013**	0.028

where $L_{A^*}(c_{start}, c_{goal})$ is the length of the shortest path between the start cell and the goal cell, which is calculated with A* algorithm by only considering the static obstacles. This metric indicates the percentage of detour w.r.t the shortest path length.

- *Computing Time*: This measure corresponds to the average computing time at each step during the testing.

5.4.4 Results

In this section, we present comparative results for both the single-robot and multi-robot path planning tasks.

Single-Robot Path Planning Results

We compare our approach with dynamic A* based methods with global re-planning and local re-planning strategies, and we call these two methods `Global Re-planning` and `Local Re-planning` respectively. For `Global Re-planning`, each time the robot encounters a conflict, an alternative path is searched for from the current cell to the goal cell by using the A* method, considering the current position of all the dynamic obstacles. For `Local Re-planning`, an alternative path is searched for from the current cell to the farthest cell within the robot's FOV. We also compare our reward function with a naive reward function, which strictly encourages the robot to follow the global guidance. Concretely, if the robot's next location is on the global guidance, we give a constant positive reward $R(t) = 0.01$. Otherwise, we give a large negative reward $R(t) = r_1 + D_r \times r_4$, where $r_4 = -0.03$ and D_r is computed by calculating the distance between the robot's next location with the nearest global guidance cell. We set a time-out for all the tests, within which time if the robot can

not reach its goal, this test is defined as a failure case. In each test, the time-out value is set as double of the Manhattan distance between the start cell and the goal cell of the robot. We train our model and the naive reward-based model by using both the three environments shown in Figure 5.8, and then compare our testing results with `Global Re-planning` and `Local Re-planning` in the three environments separately. For each map, we separate the comparison into three groups with different Manhattan distances between the start cell and the goal cell, which are set to 50, 100, and 150, respectively. For each group, 100 pairs of start and goal locations are randomly selected and the mean value and its standard deviation are calculated. The desired trajectories of dynamic obstacles are consistent among the testing of each method for a fair comparison.

The comparison results in Table 5.1 validate that: 1) Compared with `Local Re-planning` and `Global Re-planning`, our approach uses the smallest number of moving steps in all the cases. 2) Our approach has the smallest standard deviations in all the cases, which demonstrates our consistency across different settings. 3) Since under the naive reward function, the robot is encouraged to follow the global guidance strictly, its performance is much worse than when utilizing our reward function. The naive reward introduces more detour steps and waiting time steps in the presence of motion conflicts, and may even cause deadlocks (which further lead to failure cases). In contrast, our reward function ensures the convergence of the navigation tasks while simultaneously encouraging the robot to explore all the potential solutions to reach the goal cell with the minimum number of steps. Please note that the `Moving Cost` and `Detour Percentage` are calculated by only considering the successful cases. The range of success rate of the naive reward based approach is between $68\% - 89\%$, whereas for ours it is consistently 100%. 4) Our computing time is slightly higher than `Local Re-planning` and `Global Re-planning`, but still remains within an acceptable interval. Our maximum computing time is about $28ms$, which means that our RL planner achieves an update frequency of more than $35Hz$ on the given CPU platform, fulfilling the real-time requirements of most application scenarios.

Table 5.2 Ablation study on different robot FOV sizes. Values are the mean values of the Moving Cost index.

$H_l \times W_l$	7×7	9×9	11×11	13×13	15×15
Random-50	1.49	1.35	1.32	1.24	1.21
Random-100	1.33	1.25	1.23	1.17	1.15
Random-150	1.27	1.21	1.18	1.14	1.13

Table 5.3 Ablation study on different input sequence lengths. Values are the mean values of the Moving Cost index.

N_t	1	2	3	4
Random-50	1.42	1.28	1.22	1.21
Random-100	1.48	1.34	1.21	1.15
Random-150	1.56	1.31	1.19	1.13

We further test the effect of different robot FOV size and input sequence length N_t on the performance. The random map in Figure 5.8 is used, and the start and goal cells are generated with fixed Manhattan distances of 50, 100, and 150. For each value, 100 pairs of start and goal cells are tested. We first choose different values of the FOV size of $H_l \times W_l$ for comparison. The results in Table 5.2 show that: 1) As the FOV size increases, the robot reaches the goal cell in less average steps. 2) The performance improvement is not significant when the FOV size is large than 13×13. Since a smaller FOV size implies a smaller learning model, a FOV size of 15×15 is large enough for our cases to balance the performance and computation cost. We then compare different values of the input sequence length N_t for comparison. For ease of implementation, we keep the same network input size in all the cases and use empty observation images for the cases with $N_t < 4$. Note that if $N_t = 1$, the robot loses all the temporal information of the dynamic obstacles and only considers the current observation. The results in Table 5.3 show that: 1) Introducing the historical local observation information improves the system performance in all the cases significantly. 2) When $N_t > 3$, increasing N_t only marginally improves the performance. Since a smaller N_t implies a smaller learning model, $N_t = 4$ is large enough for our cases to balance the performance and computation cost.

Table 5.4 Experiment results in unseen environments

	Local	Global	Ours
Moving Cost	1.46(0.19)	1.21(0.08)	**1.12(0.06)**
Detour Percentage	42.8(19)%	20.5(8)%	**9.6(6)**%

Next, we test our approach in an unseen environment to validate its generalizability. The environment is an enlarged version of the random map in Figure 5.8 with a size 200×200. The densities of static and dynamic obstacles are set to 0.15 and 0.05, respectively. We randomly generate 100 pairs of start and goal cells with a constant Manhattan distance of 200. Note that we directly use the model trained in the small maps for testing. The results in

Table 5.5 Multi-robot path planning results: success rate and computational time of different approaches

	Success Rate						Computing Time (ms)		
	ECBS	HCA*	Global Re-planning	Discrete-ORCA	PRIMAL	Ours	Discrete-ORCA	PRIMAL	Ours
Regular	**100%**	**100%**	95.7%	88.7%	92.3%	99.7%	**2.064(0.107)**	5.892(0.104)	6.367(0.337)
Random	**100%**	**100%**	98.2%	55.0%	80.6%	99.7%	**1.233(0.044)**	6.620(0.280)	6.206(0.260)
Free	**100%**	**100%**	98.8%	99.5%	75.7%	99.8%	**1.480(0.050)**	7.319(0.300)	6.246(0.277)

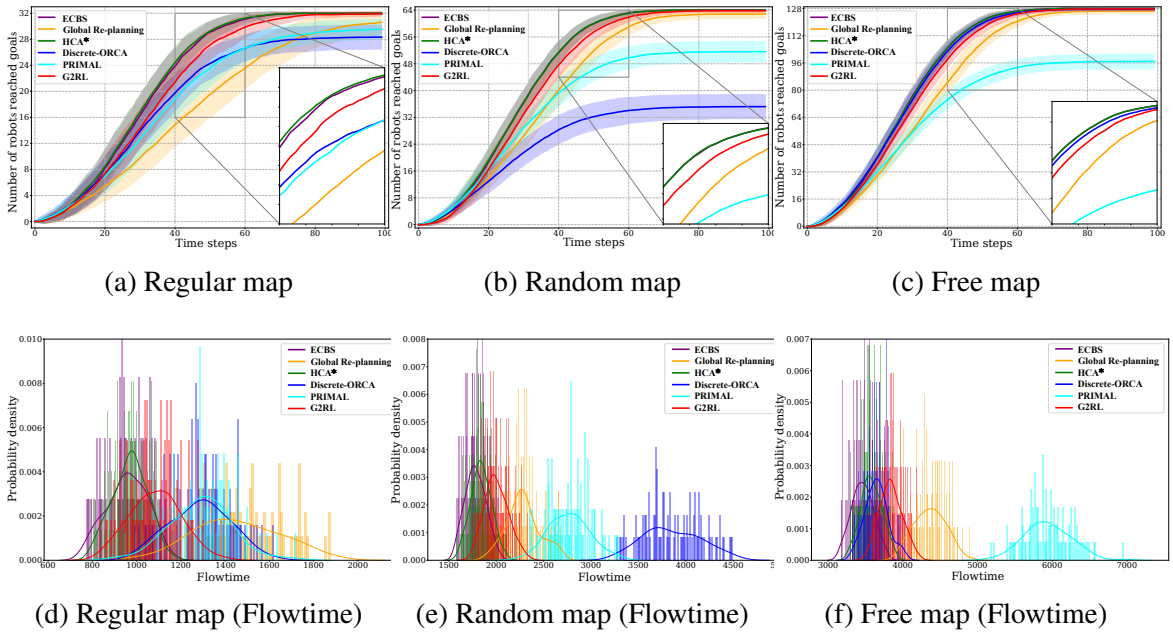
Table 5.1 and Table 5.4 show that our approach performs consistently well, under different environments, which validates the scalability and generalizability of our approach.

Comparison with Multi-Robot Path Planning Methods

We apply our method to the problem of multi-robot path planning, and compare it to five state-of-the-art benchmarks: *(i)* a decoupled dynamic A* based method, `Global Re-planning`, *(ii)* a decoupled path planning method, `HCA*`[10], *(iii)* a centralized optimal path planning method, Conflict-Based Search (`CBS`)[14], *(iv)* a velocity-based method `ORCA`[45] and, *(v)*, the RL based approach `PRIMAL`[15]. For `HCA*`, the priorities of the robots are randomly chosen. For `CBS`, since computing an optimal solution when the robot number is larger than 50 is often intractable, we use its sub-optimal version `ECBS` [14] with a sub-optimality bound set to 1.03. For `ORCA`, we calculate the next position of the robot by only considering the angle of the velocity output and thus transform the continuous-space `ORCA` into a discrete version `Discrete-ORCA`. It should be noted that neither our approach nor `PRIMAL` has been trained in the testing maps. For `PRIMAL`, we directly use the trained model provided online by the authors [3]. In our approach, we directly use the model trained in a single robot case.

Comparisons are performed in three different maps with size 40×40 and varying numbers of robots, which are smaller versions of the maps in Figure 5.9. The static obstacle densities are set to 0.45, 0.15, and 0 in the regular, random, and free maps, respectively, and the robot numbers are set to 32, 64, and 128. In each map, we generate 100 random configurations of robots with different start and goal cells. To ensure the solvability of the problem, once each robot arrives at its goal cell, we remove the robot from the environment to avoid conflicts. We set a time-out of 100 time-steps for all the tests, within which time, if any robot can not reach its goal, this test is defined as a failure case. In Figures 5.9 (a)-(c), we compare the number of robots that have reached their goals as a function of time. In Figures 5.9 (d)-(f), we compare the flowtime of the different approaches, which is defined as the sum of traversal time steps across all the robots involved in the instance. In Table 5.5, we compare their success rates and computing times. Since in `Global Re-planning`, `ECBS` and `HCA*`, the robot action is

[3] https://github.com/gsartoretti/distributedRL_MAPF



(a) Regular map (b) Random map (c) Free map

(d) Regular map (Flowtime) (e) Random map (Flowtime) (f) Free map (Flowtime)

Fig. 5.9 Multi-robot path planning results. The upper row shows the number of reached robots at different steps of different approaches in three testing maps. The solid lines show the average number across 100 tests, and the shadow areas represent the standard deviations. The lower row plots the corresponding histograms of flowtime values. The flowtime of all the failure cases is set to the maximum time step 100.

not generated online at each step, we only compare the computing time of `Discrete-ORCA`, `PRIMAL` and our approach. The computing time is normalized by the average flowtime.

These results show that: 1) Our approach maintains consistent performance across different environments, outperforming `Global Re-planning` and `PRIMAL`, also outperforming `Discrete-ORCA` in most cases (`Discrete-ORCA` can not handle the crowded static obstacles, so it is only effective in the free map). `ECBS` and `HCA`* achieve the best performance overall, since they are both centralized approaches that have access to all robots' trajectory information. In `HCA`*, the trajectory information of all robots with higher priorities is shared and utilized in the planning of the robots with lower priorities. Our approach is in stark contrast to these approaches, since it is fully distributed and non-communicative (i.e., requiring no trajectory information of other robots). 2) Our success rate is similar to that of `ECBS` and `HCA`*, and higher than `Global Re-planning`, `Discrete-ORCA` and `PRIMAL`. The results show that our approach outperforms all the distributed methods, which validates its robustness, scalability, and generalizability. We note that in contrast to `PRIMAL`, which uses a general 'direction vector'[15], we utilize the global guidance instead. The global guidance provides dense global information which considers all the static obstacles. Thus, our method is able to overcome many of the scenarios (e.g., deadlocks) where `PRIMAL` gets stuck.

5.5 Conclusion

In this chapter, we explored two approaches to maintain the interpretability and guarantee bounded optimality of solutions when deploying a machine learning approach to solving path planning problems.

Firstly, we proposed the Graph Transformer as a heuristic function to accelerate CBS by guiding the tree search toward promising nodes with feasible solutions. While achieving significant acceleration, our approach guarantees provable completeness and bounded-suboptimality. We show that in two continuous environments with dense motion graphs, our method outperforms three classical MAPF baselines (CBS, ECBS, and ORCA), while generalizing to unseen tests with a higher number of agents remarkably well.

Secondly, we introduced G2RL, a hierarchical path-planning approach that enables end-to-end learning with a fixed-sized model in arbitrarily large environments. We provided an application of our approach to distributed uncoupled multi-robot path planning that is naturally scaled to an arbitrary number of robots. Compared with traditional path planning approaches, our proposed method is able to exploit both spatial and temporal information from raw sensory observations, which avoids unnecessary detours in large complex environments. Compared with other learning-based approaches, our proposed method has the advantage of being able to keep a fixed learning model in arbitrarily large environments. It avoids over-fitting as the map grows larger and reduces the computation cost. Experimental results validated the robustness, scalability, and generalizability of this path-planning approach. Notably, we demonstrated that its application to multi-robot path planning outperforms existing distributed methods and that it performs similarly to state-of-the-art centralized approaches that assume global dynamic knowledge.

Chapter 6

Conclusion and Future Work

In this thesis, I address *(i)* how to use GNN-based decentralized approaches to compute near-optimal solutions fast, *(ii)* how we build a ROS2-based decentralized system to port our solution to reality, *(iii)* how to generate a bounded-optimality solution or interpret the solution. These three components can also be broken into four key topics, as they are central to the learning process: expert and data generation, communication strategies for decentralized control, sim-to-real transfer, and bounded optimality and interpretability. We will discuss them in detail as follows.

6.1 Experts and Data Generation

How to generate expert data? In the Section 'Expert Data Generation' from Chapter. 3, we showed that it is possible to train decentralized controllers to learn communication and action policies that optimize a global objective by imitating a centralized optimal algorithm (CBS) as the expert. The former work considered the specific case-study of multi-agent path planning, and used Conflict-Based Search (CBS) [196] to find optimal solutions (i.e., sets of optimal, collision-free paths). CBS was deployed offline to generate sets of optimal, collision-free paths, and used it only during the training stage. Although their results demonstrated unprecedented performance in decentralized systems (i.e., achieving higher than 96% success rates with single-digit flowtime increases, compared to the expert solution), but observed poor generalization. Simply training the models through behavior cloning leads to bias and over-fitting, since the performance of the network is intrinsically constrained by the dataset. Alternative approaches include learning curricula [197] to optimize the usage of the existing training set, or the introduction of data augmentation mechanisms, which allow experts to teach the learner how to recover from past mistakes.

How to augment existing datasets? One of the major limitations of behavior cloning is that it does not learn to recover from failures, and is unable to handle unseen situations [198]. For example, if the policy has deviated from the optimal trajectory at one-time step, it will fail in getting back to states seen by the expert, hence, resulting in a cascade of errors. One solution (i.e., DAgger [112]) is to introduce the expert *during* training to teach the learner how to recover from past mistakes.

In Section 'Expert Data Generation' from Chapter. 3, we demonstrated the utility of this approach by making use of a novel dataset aggregation method that leverages an online expert to resolve hard cases during training. Other approaches are to directly extract a policy from training data, such as GAIL [199]. More broadly speaking, with data augmentation, one can produce arbitrary amounts of training data from arbitrary probability distributions to account for a variety of factors, such as roadmap structure, local environment, obstacle density, motion characteristics, and local robot configurations. Such carefully controlled distributions enable us to introduce different levels of local coordination difficulties and generate the most challenging instances at each training stage, inherently achieving a form of curriculum learning. In addition, data augmentation allows us to understand the ability boundary of the trained model, to analyze the correlation between different factors, and to find identify factors that have the strongest effect on the system performance.

6.2 Communication Strategies for Decentralized Control

What, how and when to send information? While effective communication is key to decentralized control, it is far from obvious *what information is crucial to the task, and what must be shared among agents.* This question differs from problem to problem and the optimal strategy is often unknown. Hand-engineered coordination strategies often fail to deliver the desired performance, and despite ongoing progress in this domain, they still require substantial design effort. Recent work has shown the promise of Graph Neural Networks (GNNs) to learn explicit communication strategies that enable complex multi-agent coordination [118, 117, 18, 19]. In the context of multi-robot systems, individual robots are modeled as nodes, the communication links between them as edges, and the internal state of each robot as graph signals. By sending messages over the communication links, each robot in the graph indirectly receives access to the global state. The key attribute of GNNs is that they compress data as it flows through the communication graph. In effect, this compresses the global state, affording agents access to global data without inundating them with the entire raw global state. Since compression is performed on local networks (with parameters that can be shared across the entire graph), GNNs are able to compress previously

unseen global states. In the process of learning how to compress the global state, GNNs also learn which elements of the signal are the most important, and discard the irrelevant information [118]. This produces a non-injective mapping from global states to latent states, where similar global states 'overlap', further improving generalization. SchedNet[200] was proposed based on a multi-agent deep reinforcement learning framework that enables agents to schedule communication, encode messages, and select actions based on received messages

Are all messages equally important? Unfortunately, if communication happens concurrently and equivalently among many neighboring robots, it is likely to cause redundant information, burden the computational capacity and adversely affect overall team performance. Hence, new approaches towards *communication-aware planning* are required. A potential approach is to introduce *attention mechanisms* to actively measure the relative importance of messages (and their senders). Attention mechanisms have been actively studied and widely adopted in various learning-based models [114], which can be viewed as dynamically amplifying or reducing the weights of features based on their relative importance computed by a given mechanism. Hence, the network can be trained to focus on task-relevant parts of the graph [107].

In Sec. 3.5 from Chapter. 3, we integrated an attention mechanism with a GNN-based communication strategy to allow for *message-dependent attention* in a multi-agent path planning problem. A key-query-like mechanism determines the relative importance of features in the messages received from various neighboring robots. Their results show that it is possible to achieve performance close to that of a coupled centralized expert algorithm, while scaling to problem instances that are $\times 100$ larger than the training instances.

6.3 Sim-to-Real Transfer

Expert data is typically generated in a simulation, yet policies trained in simulation often do not generalize to the real world. This is referred to as the *reality gap* [201].

Why is sim-to-real transfer difficult? Even though simulations have become more realistic and easily accessible over recent years [202, 203], it is computationally infeasible to replicate all aspects of real-world physics in a simulation since the uncertainty and randomness of complex robot-world interactions are difficult to model. Domain randomization is an intuitive solution to this problem, but also makes the task of learning harder than necessary and therefore results in sub-optimal policies. While the reality gap is a major challenge in computer vision, robotics also deals with the physical interaction with the real world and

physical constraints such as inertia, for example in robotic grasping [155, 204], drone flight [205, 206] or robotic locomotion [207, 208].

Why is sim-to-real transfer even more difficult for multi-robot systems? While sim-to-real in the single-robot domain typically deals with robot-world interaction, the multi-robot domain is also concerned with robot-robot interactions. An example of this is a swarm of drones flying closely to each other and turbulence affecting the motions of other drones in the vicinity. We already have established that communication is key to efficient multi-robot interaction, but it is not obvious how such communications are affected by the reality gap. Multi-robot coordination is typically trained in a synchronous manner, but when deploying these policies to the real-world, decentralized communication is *asynchronous*. Furthermore, randomness such as message dropouts and delays are typically not considered during synchronous training. To the best of our knowledge, no research has been conducted that evaluates those factors and the impact they have on the performance of policies. Decentralization is key to successful multi-agent systems, therefore decentralized mesh communication networks are required to operate multi-robot systems in the real world, which may pose additional challenges to the sim-to-real transfer. Lastly, during cooperative training it is typically assumed that all agents are being truthful about their communications, but faulty and malicious agents can be part of the real world and cause additional problems [37, 209].

How do we build up decentralized multi-robot systems? Even though GNNs have an inherently decentralizable mathematical formulation, previous work on GNN-based multi-robot policies was conducted exclusively in centralized simulations using synchronous communication [19, 118, 18]. To port our solution into reality, in Sec. 4.3 from Chapter 4, we developed a ROS2-based system that allows for the fully decentralized execution of GNN-based policies. With this, the policies derived through GNN-based learning schemes can be deployed to the real world on physical multi-robot systems. We demonstrated our framework on a case study that requires tight coordination between robots, and presented first-of-a-kind results that showed the successful real-world deployment of GNN-based policies on a decentralized multi-robot system relying on Adhoc communication. In the case study of this work, we used domain randomization to facilitate the process of making the real-world a permutation of the training environment as a simple solution before deploying the trained policy into real-world. Yet, it led to an observable sim-to-real gap.

How can we close the reality gap? In Sec. 4.4 from Chapter. 4, we proposed a vision-only-based learning approach that leverages a GNN to encode and communicate relevant

viewpoint information to the mobile robot. During navigation, the robot is guided by a model that we train through imitation learning to approximate optimal motion primitives, thereby predicting the effective cost-to-go (to the target). Our experiments demonstrated its generalizability in guiding robots in previously unseen environments with various sensor layouts.

We see a few possible avenues to tackle the sim-to-real transfer for multi-robot communication. More realistic (network) simulations [120] are always helpful, but also costly alternatives. Methods such as *sim-to-real via real-to-sim* [210] or training agents in the real-world in a *mixed reality* setting [211] and federated, decentralized learning where individual robots collect data and use it to update a local model that is then aggregated into a global model can benefit the sim-to-real transfer [212, 213]. Federated learning allows each robot to keep its local data private, as the training process occurs locally without sharing the raw data with a central server. Federated learning can accommodate large-scale multi-robot systems since the training process is distributed across multiple robots. This enables efficient utilization of computational resources and can handle a growing number of robots. Each robot in a decentralized multi-robot system may have different sensory inputs, environmental conditions, or task requirements. Federated learning allows individual robots to adapt their local models based on their unique experiences, enhancing the system's overall performance and versatility.

6.4 Bounded-suboptimality and Interpretability

Vanilla GNN-based decentralized path planning has demonstrated its performance empirically via an end-to-end learning approach. However, these black box approaches are facing challenges to directly deploy in the actual workplace, as they are hard to find a guaranteed and interpretable solution.

How to generate a bounded-suboptimal solution? In Sec. 5.3, we designed Graph Transformer, as a heuristic function, to accelerate the focal search within Conflict-Based Search (CBS) in a non-grid setting, especially dense graphs. Our framework guarantees both the completeness and bounded suboptimality of the solution. This approach also can be our first attempt to deploy our method from a discrete into a continuous domain.

How to make the reward mechanism in RL interpretable? Re-planning strategies are commonly used to cope with dynamic obstacles, where a planning algorithm searches for an alternative path whenever the robot encounters a conflict. To avoid the unnecessary detours

caused by re-planning strategies, For the explainability and interpretability of reinforcement learning, in Sec. 5.3, we introduced a globally Guided Reinforcement Learning approach (G2RL), which first utilizes a global path planning algorithm (for example, A*) to obtain a globally optimal path, called the global guidance. This novel reward structure provides not only the interpretability of our framework but also dense rewards. It does not require the robot to strictly follow global guidance at every step, thus encouraging the robot to explore all potential solutions. As our reward function is independent of the environment, our trained framework generalizes to arbitrary environments and can be used to solve the multi-robot path planning problem in a fully distributed reactive manner.

6.5 Future Avenues

The sections above lay out the challenges entailed by the described approach. Yet, this begs the following two questions:

Is imitation learning the right paradigm? There are two main approaches to training a controller for a multi-robot system: imitation learning (e.g., [214]) and reinforcement learning (e.g., [215]). The most obvious benefit to RL is that it does not require an expert algorithm, as it simply optimizes a reward. However, the reward function requires careful consideration to guarantee that the learned controller does not exploit it by using unsafe or inappropriate actions. Conversely, IL is often biased around regions that can be reached by the expert and, consequently, if the controller ever finds itself in a previously unseen situation, it might exhibit unpredictable behavior. Finally, IL is inherently limited by the expert algorithm. As such, possible future directions should explore the combination of both IL and RL (e.g., [36]) in the context of decentralized multi-robot systems.

Is it possible to learn small-scale coordination patterns for large-scale systems? Ideally, we hope that controllers trained on only a few robots (which not only facilitates data generation, but also accelerates the training process), can then be deployed on large-scale systems with hundreds and even thousands of robots. Achieving this expectation may be within our reach. A recent example can be found in [77], where the local coordination behaviors and conventions learned in a partially observable world successfully scales up to 2048 mobile robots in crowded and highly-structured environments. In Sec. 3.5 from Chapter. 3, a promising demonstration shows that a policy trained in 20×20 maps with only 10 robots obtains a success rate above 80% in 200×200 maps with 1000 robots in different environments with 10% obstacle density, and more impressively, the learned policy only

spends $\frac{1}{30}$ computation time compared to the centralized expert. Overall, these preliminary results give us confidence that we should continue leveraging methods, such as IL, to distill offline-optimal algorithms to online-scalable controllers.

References

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

[2] Binyu Wang, Zhe Liu, Qingbiao Li, and Amanda Prorok. Mobile robot path planning in dynamic environments through globally guided reinforcement learning. *arXiv preprint arXiv:2005.05420*, 2020.

[3] Amanda Prorok and Vijay Kumar. Privacy-preserving vehicle assignment for mobility-on-demand systems. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 1869–1876. IEEE, 2017.

[4] Jeroen Ploeg, Bart TM Scheepers, Ellen Van Nunen, Nathan Van de Wouw, and Henk Nijmeijer. Design and experimental evaluation of cooperative adaptive cruise control. In *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*, pages 260–265. IEEE, 2011.

[5] CNET. Cnet news - meet the robots making amazon even faster, 2014. https://www.youtube.com/watch?v=UtBa9yVZBJM, Last accessed on 2019-02-14.

[6] Social Development. Population Division. *World Urbanization Prospects: The 2001 Revision*, volume 216. United Nations Publications, 2002.

[7] John Enright and Peter R Wurman. Optimization and coordinated autonomy in mobile fulfillment systems. In *Automated Action Planning for Autonomous Mobile Robots*, pages 33–38, 2011.

[8] Luis Merino, Fernando Caballero, J Ramiro Martínez-de Dios, Joaquin Ferruz, and Aníbal Ollero. A cooperative perception system for multiple uavs: Application to automatic detection of forest fires. *Journal of Field Robotics*, 23(3-4):165–184, 2006.

[9] Gerald Steinbauer and Alexander Ferrein. 20 years of robocup. *KI-Künstliche Intelligenz*, 30:221–224, 2016.

[10] David Silver. Cooperative pathfinding. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 117–122, 2005.

[11] Trevor Standley and Richard Korf. Complete algorithms for cooperative pathfinding problems. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pages 668–673, Barcelona, Catalonia, Spain, 2011. AAAI Press.

[12] J. P. van den Berg and M. H. Overmars. Prioritized motion planning for multiple robots. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 430–435, Aug 2005.

[13] Wenying Wu, Subhrajit Bhattacharya, and Amanda Prorok. Multi-robot path deconfliction through prioritization by path prospects. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9809–9815. IEEE, 2020.

[14] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *European Conference on Artificial Intelligence*, pages 961–962, 2014.

[15] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK Kumar, Sven Koenig, and Howie Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4:2378–2385, 2019.

[16] Aravind Rajeswaran, Kendall Lowrey, Emanuel V. Todorov, and Sham M Kakade. Towards generalization and simplicity in continuous control. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pages 6550–6561. Curran Associates, Inc., 2017.

[17] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. 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)*, pages 23–30, September 2017.

[18] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro. Learning decentralized controllers for robot swarms with graph neural networks. In *Conference Robot Learning 2019*, Osaka, Japan, 30 Oct.-1 Nov. 2019.

[19] Qingbiao Li, Fernando Gama, Alejandro Ribeiro, and Amanda Prorok. Graph neural networks for decentralized multi-robot path planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 11785–11792. IEEE, 2020.

[20] Jingjin Yu and Steven LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27, pages 1443–1449, 2013.

[21] Mac Schwager, Daniela Rus, and Jean-Jacques Slotine. Decentralized, adaptive coverage control for networked robots. *The International Journal of Robotics Research*, 28(3):357–375, 2009.

[22] Ishat E Rabban and Pratap Tokekar. Improved resilient coverage maximization with multiple robots. *arXiv preprint arXiv:2007.02204*, 2020.

[23] S S Ponda, L B Johnson, and J P How. Distributed chance-constrained task allocation for autonomous multi-agent teams. In *American Control Conference (ACC)*, pages 4528–4533, 2012.

[24] Amanda Prorok. Redundant Robot Assignment on Graphs with Uncertain Edge Costs. In Nikolaus Correll, Mac Schwager, and Michael Otte, editors, *Distributed Autonomous Robotic Systems*, Springer Proceedings in Advanced Robotics, pages 313–327, 2019.

[25] Michael M. Zavlanos, Leonid Spesivtsev, and George J. Pappas. A distributed auction algorithm for the assignment problem. In *IEEE Conference on Decision and Control*, pages 1212–1217, December 2008. ISSN: 0191-2216.

[26] N Michael, M M Zavlanos, V. Kumar, and G J Pappas. Distributed multi-robot task assignment and formation control. In *IEEE International Conference Robotics and Automation*, pages 128–133, 2008.

[27] Boyoon Jung and Gaurav S Sukhatme. Cooperative multi-robot target tracking. In *Distributed autonomous robotic systems 7*, pages 81–90. Springer, 2006.

[28] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the TSP by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, pages 170–181. Springer, 2018.

[29] Benjamin Hudson, Qingbiao Li, Matthew Malencia, and Amanda Prorok. Graph neural network guided local search for the traveling salesperson problem. *International Conference on Learning Representations (ICLR)*, 2021.

[30] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018.

[31] Elias B Khalil, Bistra Dilkina, George L Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 659–666, 2017.

[32] Sammie Katt, Frans A Oliehoek, and Christopher Amato. Learning in pomdps with monte carlo tree search. In *International Conference on Machine Learning*, pages 1819–1827. PMLR, 2017.

[33] Charles Richter, John Ware, and Nicholas Roy. High-speed autonomous navigation of unknown environments using learned probabilities of collision. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6114–6121. IEEE, 2014.

[34] Katherine Liu, Martina Stadler, and Nicholas Roy. Learned sampling distributions for efficient planning in hybrid geometric and object-level representations. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9555–9562. IEEE, 2020.

[35] Vishnu R Desaraju and Jonathan P How. Decentralized path planning for multi-agent teams with complex constraints. *Autonomous Robots*, 32(4):385–403, 2012. Publisher: Springer.

[36] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P Agapiou, Max Jaderberg, Alexander S Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. In *Nature*, 2019.

[37] Jan Blumenkamp and Amanda Prorok. The emergence of adversarial communication in multi-agent reinforcement learning. *Conference on Robot Learning (CoRL)*, 2020.

[38] Jiankun Wang, Tianyi Zhang, Nachuan Ma, Zhaoting Li, Han Ma, Fei Meng, and Max Q-H Meng. A survey of learning-based robot motion planning. *IET Cyber-Systems and Robotics*, 3(4):302–314, 2021.

[39] Qingbiao Li, Weizhe Lin, Zhe Liu, and Amanda Prorok. Message-aware graph attention networks for large-scale multi-robot path planning. *IEEE Robotics and Automation Letters*, 6(3):5533–5540, 2021.

[40] Lifeng Zhou, Vishnu D Sharma, Qingbiao Li, Amanda Prorok, Alejandro Ribeiro, Pratap Tokekar, and Vijay Kumar. Graph neural networks for decentralized multi-robot target tracking. In *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 195–202. IEEE, 2022.

[41] Jan Blumenkamp, Qingbiao Li, Binyu Wang, Zhe Liu, and Amanda Prorok. See what the robot can't see: Learning cooperative perception for visual navigation. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2023.

[42] Jan Blumenkamp, Steven Morad, Jennifer Gielis, Qingbiao Li, and Amanda Prorok. A framework for real-world multi-robot systems running decentralized gnn-based policies. *IEEE International Conference on Robotics and Automation*, 2021.

[43] Friedemann Mattern. Algorithms for distributed termination detection. *Distributed computing*, 2(3):161–175, 1987.

[44] Stephen J Guy, Jatin Chhugani, Changkyu Kim, Nadathur Satish, Ming Lin, Dinesh Manocha, and Pradeep Dubey. Clearpath: highly parallel collision avoidance for multi-agent simulation. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 177–187. ACM, 2009.

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

[46] Amichai Levy, Chris Keitel, Sam Engel, and James McLurkin. The extended velocity obstacle and applying orca in the real world. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 16–22. IEEE, 2015.

[47] John E Hopcroft, Jacob Theodore Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; pspace-hardness of the" warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88, 1984.

[48] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

[49] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[50] Glenn Wagner and Howie Choset. M*: A complete multirobot path planning algorithm with performance bounds. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3260–3267. IEEE, 2011.

[51] Cornelia Ferner, Glenn Wagner, and Howie Choset. Odrm* optimal multirobot path planning in low dimensional search spaces. In *2013 IEEE International Conference on Robotics and Automation*, pages 3854–3859. IEEE, 2013.

[52] Raz Nissim and Ronen Brafman. Distributed heuristic forward search for multi-agent planning. *Journal of Artificial Intelligence Research*, 51:293–332, 2014.

[53] Maren Bennewitz, Wolfram Burgard, and Sebastian Thrun. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and autonomous systems*, 41(2-3):89–99, 2002.

[54] Prasanna Velagapudi, Katia Sycara, and Paul Scerri. Decentralized prioritized planning in large multirobot teams. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4603–4609. IEEE, 2010.

[55] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[56] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Meta-agent conflict-based search for optimal multi-agent path finding. *SoCS*, 1:39–40, 2012.

[57] Ariel Felner, Jiaoyang Li, Eli Boyarski, Hang Ma, Liron Cohen, TK Satish Kumar, and Sven Koenig. Adding heuristics to conflict-based search for multi-agent path finding. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*, 2018.

[58] Jiaoyang Li, Zhe Chen, Daniel Harabor, P Stuckey, and Sven Koenig. Anytime multi-agent path finding via large neighborhood search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[59] Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J Stuckey, and Sven Koenig. Mapf-lns2: fast repairing for multi-agent path finding via large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10256–10265, 2022.

[60] Sergio Caccamo, Ramviyas Parasuraman, Luigi Freda, Mario Gianni, and Petter Ögren. Rcamp: A resilient communication-aware motion planner for mobile robots with autonomous repair of wireless connectivity. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2010–2017. IEEE, 2017.

[61] Md Mahbubur Rahman, Leonardo Bobadilla, Franklin Abodo, and Brian Rapp. Relay vehicle formations for optimizing communication quality in robot networks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6633–6639. IEEE, 2017.

[62] Graeme Best, Michael Forrai, Ramgopal R Mettu, and Robert Fitch. Planning-aware communication for decentralised multi-robot coordination. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 21, 2018.

[63] Michael Fowler, Pratap Tokekar, T Charles Clancy, and Ryan K Williams. Constrained-action pomdps for multi-agent intelligent knowledge distribution. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[64] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[65] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[66] Mohak Bhardwaj, Sanjiban Choudhury, and Sebastian Scherer. Learning heuristic search via imitation. In *Conference on Robot Learning*, pages 271–280. PMLR, 2017.

[67] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[68] Binghong Chen, Bo Dai, and Le Song. Learning to plan via neural exploration-exploitation trees. *International Conference on Learning Representations (ICLR)*, 2019.

[69] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[70] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3067–3074. IEEE, 2015.

[71] Brian Ichter, James Harrison, and MarcoShen Pavone. Learning sampling distributions for robot motion planning. In *IEEE International Conference on Robotics and Automation*, pages 7087–7094, 2018.

[72] Mayur J Bency, Ahmed H Qureshi, and Michael C Yip. Neural path planning: Fixed time, near-optimal path generation via oracle imitation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3965–3972. IEEE, 2019.

[73] Stefan Schaal, Auke Ijspeert, and Aude Billard. Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1431):537–547, 2003.

[74] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[75] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[76] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *IEEE International Conference on Robotics and Automation*, pages 285–292. IEEE, 2017.

[77] Mehul Damani, Zhiyao Luo, Emerson Wenzel, and Guillaume Sartoretti. Primal$_2$: Pathfinding via reinforcement and imitation multi-agent learning - lifelong. *IEEE Robotics and Automation Letters*, 6(2):2666–2673, 2021.

[78] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.

[79] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.

[80] Peng Peng, Ying Wen, Yaodong Yang, Quan Yuan, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2017.

[81] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

[82] Yutong Wang and Guillaume Sartoretti. Fcmnet: Full communication memory net for team-level cooperation in multi-agent systems. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2022.

[83] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[84] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[85] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. Heterogeneous network embedding via deep architectures. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 119–128. ACM, 2015.

[86] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems*, pages 6530–6539, 2017.

[87] Aleksandra Malysheva, Tegg Taekyong Sung, Chae-Bong Sohn, Daniel Kudenko, and Aleksei Shpilman. Deep multi-agent reinforcement learning with relevance graphs. *arXiv preprint arXiv:1811.12557*, 2018.

[88] Fernando Gama, Antonio G Marques, Geert Leus, and Alejandro Ribeiro. Convolutional neural network architectures for signals supported on graphs. *IEEE Transactions on Signal Processing*, 67(4):1034–1049, 2018.

[89] Laëtitia Matignon, Laurent Jeanpierre, and Abdel-Illah Mouaddib. Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes. In *Twenty-sixth AAAI conference on artificial intelligence*, 2012.

[90] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[91] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro. Convolutional neural network architectures for signals supported on graphs. *IEEE Trans. Signal Process.*, 67(4):1034–1049, Feb. 2019.

[92] Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, page 1443–1449. AAAI Press, 2013.

[93] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[94] Cornelia Ferner, Glenn Wagner, and Howie Choset. ODrM*: Optimal multirobot path planning in low dimensional search spaces. In *IEEE International Conference on Robotics and Automation*, pages 3854–3859, 2013.

[95] Vishnu R Desaraju and Jonathan P How. Decentralized path planning for multi-agent teams with complex constraints. *Autonomous Robots*, 32(4):385–403, 2012.

[96] Y. Wang, N. Matni, and J. C. Doyle. Separable and Localized System-Level Synthesis for Large-Scale Systems. *IEEE Transactions on Automatic Control*, 63(12):4234–4249, December 2018.

[97] M. Rotkowitz and S. Lall. A characterization of convex problems in decentralized control. *IEEE Transactions on Automatic Control*, 50(12):1984–1996, December 2005.

[98] Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham Kakade. Towards generalization and simplicity in continuous control. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 6553–6564, Red Hook, NY, USA, 2017. Curran Associates Inc.

[99] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 23–30, September 2017.

[100] M. Everett, Y. F. Chen, and J. P. How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3052–3059, October 2018.

[101] Arbaaz Khan, Ekaterina Tolstaya, Alejandro Ribeiro, and Vijay Kumar. Graph policy gradients for large scale robot control. In *Conference on robot learning*, pages 823–834. PMLR, 2020.

[102] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[103] Amanda Prorok. Graph Neural Networks for Learning Robot Team Coordination. *Federated AI for Robotics Workshop, IJCAI-ECAI/ICML/AAMAS 2018; arXiv:1805.03737 [cs]*, May 2018. arXiv: 1805.03737.

[104] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pages 2224–2232, 2015.

[105] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272, 2017.

[106] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and others. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pages 4502–4510, 2016.

[107] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *International Conference on Learning Representations (ICLR)*, 2017.

[108] Yen-Cheng Liu, Junjiao Tian, Nathaniel Glaser, and Zsolt Kira. When2com: multi-agent perception via communication graph grouping. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4106–4115, 2020.

[109] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst. Graph signal processing: Overview, challenges and applications. *Proc. IEEE*, 106(5):808–828, May 2018.

[110] Fernando Gama, Joan Bruna, and Alejandro Ribeiro. Stability properties of graph neural networks. *IEEE Transactions on Signal Processing*, 68:5680–5695, 2020.

[111] Mehak Raibail, Abdul Hadi Abd Rahman, Ghassan Jasim AL-Anizy, Mohammad Faidzul Nasrudin, Mohd Shahrul Mohd Nadzir, Nor Mohd Razif Noraini, and Tan Siok Yee. Decentralized multi-robot collision avoidance: A systematic review from 2015 to 2021. *Symmetry*, 14(3):610, 2022.

[112] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 627–635, Fort Lauderdale, FL, USA, 2011.

[113] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, pages 1–4, Long Beach, CA, USA, 2017. Advances in Neural Information Processing Systems.

[114] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[115] Elvin Isufi, Fernando Gama, and Alejandro Ribeiro. Edgenets: Edge varying graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7457–7473, 2021.

[116] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI Conference on Artificial Intelligence*, pages 4278–4284, 2017.

[117] Arbaaz Khan, Ekaterina Tolstaya, Alejandro Ribeiro, and Vijay Kumar. Graph policy gradients for large scale robot control. In *Conference on Robot Learning*, pages 823–834, 2020.

[118] Ryan Kortvelesy and Amanda Prorok. Modgnn: Expert policy approximation in multi-agent systems with a modular graph neural network architecture. *International Conference on Robotics and Automation (ICRA)*, 2021.

[119] Steven D Morad, Stephan Liwicki, Ryan Kortvelesy, Roberto Mecca, and Amanda Prorok. Graph convolutional memory using topological priors. *International Conference on Learning Representations*, 2021.

[120] Miguel Calvo-Fullana, Daniel Mox, Alexander Pyattaev, Jonathan Fink, Vijay Kumar, and Alejandro Ribeiro. Ros-netsim: A framework for the integration of robotic and network simulators. *IEEE Robotics and Automation Letters*, 6(2):1120–1127, 2021.

[121] Jason M Gregory, Jeffrey N Twigg, and Jonathan R Fink. Enabling autonomous information-gathering and self-recovery behaviors in partially-known, communication-constrained environments. In *IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 92–99. IEEE, 2016.

[122] James Stephan, Jonathan Fink, Vijay Kumar, and Alejandro Ribeiro. Concurrent control of mobility and communication in multirobot systems. *IEEE Transactions on Robotics*, 33(5):1248–1254, 2017.

[123] James Stephan, Jonathan Fink, Benjamin Charrow, Alejandro Ribeiro, and Vijay Kumar. Robust routing and multi-confirmation transmission protocol for connectivity management of mobile robotic teams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3753–3760. IEEE, 2014.

[124] Peter Corke, Ron Peterson, and Daniela Rus. Networked robots: Flying robot navigation using a sensor net. In *International Symposium on Robotics Research*, pages 234–243, 2005.

[125] Zendai Kashino, Goldie Nejat, and Beno Benhabib. A hybrid strategy for target search using static and mobile sensors. *IEEE Transactions on Cybernetics*, 50(2):856–868, 2020.

[126] David Johnson, Tim Stack, Russ Fish, Daniel Montrallo Flickinger, Leigh Stoller, Robert Ricci, and Jay Lepreau. Mobile emulab: A robotic wireless and sensor network testbed. In *IEEE International Conference on Computer Communications*, pages 1–12. IEEE, 2006.

[127] James A. Preiss, Wolfgang Honig, Gaurav S. Sukhatme, and Nora Ayanian. Crazyswarm: A large nano-quadcopter swarm. In *IEEE International Conference on Robotics and Automation*, pages 3299–3304, 2017.

[128] Daniel Pickem, Paul Glotfelter, Li Wang, Mark Mote, Aaron Ames, Eric Feron, and Magnus Egerstedt. The robotarium: A remotely accessible swarm robotics research testbed. In *IEEE International Conference on Robotics and Automation*, pages 1699–1706. IEEE, 2017.

[129] Nathan Michael, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar. The grasp multiple micro-uav testbed. *IEEE Robotics & Automation Magazine*, 17(3):56–65, 2010.

[130] Jason A Tran, Pradipta Ghosh, Yutong Gu, Richard Kim, Daniel D'Souza, Nora Ayanian, and Bhaskar Krishnamachari. Intelligent robotic iot system (iris) testbed. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1–9. IEEE, 2018.

[131] Andrew Stubbs, Vladimeros Vladimerou, Adam Thomas Fulford, Derek King, Jeffrey Strick, and Geir E Dullerud. Multivehicle systems control over networks: a hovercraft testbed for networked and decentralized control. *IEEE Control Systems Magazine*, 26(3):56–69, 2006.

[132] Amanda Prorok, Jan Blumenkamp, Qingbiao Li, Ryan Kortvelesy, Zhe Liu, and Ethan Stump. The holy grail of multi-robot planning: Learning to generate online-scalable solutions from offline-optimal experts. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2022.

[133] Ming Zhou, Jun Luo, Julian Villella, Yaodong Yang, David Rusu, Jiayu Miao, Weinan Zhang, Montgomery Alban, Iman Fadakar, Zheng Chen, Aurora Chongxi Huang, Ying Wen, Kimia Hassanzadeh, Daniel Graves, Dong Chen, Zhengbang Zhu, Nhat Nguyen, Mohamed Elsayed, Kun Shao, Sanjeevan Ahilan, Baokuan Zhang, Jiannan Wu, Zhengang Fu, Kasra Rezaee, Peyman Yadmellat, Mohsen Rohani, Nicolas Perez Nieves, Yihan Ni, Seyedershad Banijamali, Alexander Cowen Rivers, Zheng Tian, Daniel Palenicek, Haitham bou Ammar, Hongbo Zhang, Wulong Liu, Jianye Hao, and Jun Wang. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving. In *Conference on Robot Learning*, 11 2020.

[134] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145. Curran Associates, Inc., 2016.

[135] Mengyuan Lee, Guanding Yu, and Huaiyu Dai. Decentralized inference with graph neural networks in wireless communication systems. *IEEE Transactions on Mobile Computing*, 2021.

[136] Andreas Frotzscher, Ulf Wetzker, Matthias Bauer, Markus Rentschler, Matthias Beyer, Stefan Elspass, and Henrik Klessig. Requirements and current solutions of wireless communication in industrial automation. In *IEEE International Conference on Communications Workshops*, pages 67–72, 2014.

[137] Shao-cheng Wang and Ahmed Helmy. Performance limits and analysis of contention-based IEEE 802.11 MAC. In *IEEE Conference on Local Computer Networks*, pages 418–425, 2006.

[138] Vishal Bhargava and N.S. Raghava. Improve collision in highly dense wifi environment. In *IEEE International Conference on Power Electronics, Intelligent Control and Energy Systems*, pages 1–5, 2018.

[139] Jennifer Gielis and Amanda Prorok. Improving 802.11p for delivery of safety-critical navigation information in robot-to-robot communication networks. *IEEE Communications Magazine*, 59(1):16–21, 2021.

[140] Hangil Kang, Hoyoung Kim, and Young Min Kwon. Recen:resilient manet based centralized multi robot system using mobile agent system. In *IEEE Symposium Series on Computational Intelligence*, pages 1952–1958, 2019.

[141] Tobias Kronauer, Joshwa Pohlmann, Maximilian Matthé, Till Smejkal, and Gerhard Fettweis. Latency analysis of ROS2 multi-node systems. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 1–7. IEEE, 2021.

[142] Suresh Shenoy and Jindong Tan. Simultaneous localization and mobile robot navigation in a hybrid sensor network. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1636–1641, 2005.

[143] Maxim A Batalin, Gaurav S Sukhatme, and Myron Hattig. Mobile robot navigation using a sensor network. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 636–641. IEEE, 2004.

[144] Qun Li, Michael DeRosa, and Daniela Rus. Distributed algorithms for guiding navigation across a sensor network. In *International Conference on Mobile Computing and Networking*, pages 313–325, 2003.

[145] Dhruv Shah and Leena Vachhani. Swarm aggregation without communication and global positioning. *IEEE Robotics and Automation Letters*, 4(2):886–893, 2019.

[146] Joseph Ortiz, Talfan Evans, and Andrew J. Davison. A visual introduction to gaussian belief propagation. *arXiv preprint arXiv:2107.02308*, 2021.

[147] Frank Dellaert, Richard Roberts, Varun Agrawal, Alex Cunningham, Chris Beall, Duy-Nguyen Ta, Fan Jiang, lucacarlone, nikai, Jose Luis Blanco-Claraco, Stephen Williams, ydjian, Gerry Chen, John Lambert, Akshay Krishnan, Andy Melim, Zhaoyang Lv, Jing Dong, Krunal Chande, balderdash devil, DiffDecisionTrees, Sungtae An, mpaluri, Ellon Paiva Mendes, Mike Bosse, Akash Patel, Ayush Baid, Paul Furgale, matthew-broadwaynavenio, and roderick koehle. borglab/gtsam: Release 4.2a9, January 2023.

[148] Ghulam M. Bhatti. Machine learning based localization in large-scale wireless sensor networks. *Sensors*, 18(12):4179, 2018.

[149] Piotr W. Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, L. Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. *ArXiv*, abs/1611.03673, 2017.

[150] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations*, 2017.

[151] Syeda Selina Akter and Lawrence B. Holder. Improving iot predictions through the identification of graphical features. *Sensors*, 19(15):3250, 2019.

[152] Ting-Kuei Hu, Fernando Gama, Tianlong Chen, Zhangyang Wang, Alejandro Ribeiro, and Brian M Sadler. Vgai: End-to-end learning of vision-based decentralized controllers for robot swarms. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4900–4904. IEEE, 2021.

[153] Stephane Ross and Drew Bagnell. Efficient reductions for imitation learning. In Yee Whye Teh and Mike Titterington, editors, *International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[154] Tianhao Zhang, Gregory Kahn, Sergey Levine, and P. Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *IEEE International Conference on Robotics and Automation*, pages 528–535, 2016.

[155] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12619–12629, Los Alamitos, CA, USA, jun 2019. IEEE Computer Society.

[156] Alex Church, John Lloyd, Raia Hadsell, and Nathan F. Lepora. Tactile sim-to-real policy transfer via real-to-sim image translation. In *Conference on Robot Learning*, pages 1645–1654. PMLR, 2022.

[157] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2017.

[158] Jingwei Zhang, Lei Tai, Peng Yun, Yufeng Xiong, Ming Liu, Joschka Boedecker, and Wolfram Burgard. Vr-goggles for robots: Real-to-sim domain adaptation for visual control. *IEEE Robotics and Automation Letters*, 4(2):1148–1155, 2019.

[159] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *IEEE International Conference on Computer Vision*, pages 2242–2251, 2017.

[160] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *IEEE International Conference on Robotics and Automation Workshop on Open Source Software*, volume 3, page 5. Kobe, Japan, 2009.

[161] Open Robotics. ROS 2 Design. https://design.ros2.org/, 2021. [Online; accessed 12/09/21].

[162] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 2018.

[163] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, volume 33, pages 4602–4609. AAAI Press, 2019.

[164] Omurhan A Soysal and Mehmet Serdar Guzel. An introduction to zero-shot learning: An essential review. In *International Congress on Human-Computer Interaction, Optimization and Robotic Applications*, pages 1–4. IEEE, 2020.

[165] O. Michel. Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42, 2004.

[166] Yong Koo Hwang and Narendra Ahuja. A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32, 1992.

[167] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

[168] Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

[169] Jacopo Banfi, Nicola Basilico, and Francesco Amigoni. Intractability of time-optimal multirobot path planning on 2d grid graphs with holes. *IEEE Robotics and Automation Letters*, 2(4):1941–1947, 2017.

[170] Jingjin Yu and Steven M LaValle. Planning optimal paths for multiple robots on graphs. In *IEEE International Conference on Robotics and Automation*, pages 3612–3617. IEEE, 2013.

[171] Jiaoyang Li, Wheeler Ruml, and Sven Koenig. Eecbs: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12353–12362, 2021.

[172] Taoan Huang, Sven Koenig, and Bistra Dilkina. Learning to resolve conflicts for multi-agent path finding with conflict-based search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11246–11253, 2021.

[173] Taoan Huang, Bistra Dilkina, and Sven Koenig. Learning node-selection strategies in bounded suboptimal conflict-based search for multi-agent path finding. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, 2021.

[174] Peng Li, Xinhan Huang, and Min Wang. A novel hybrid method for mobile robot path planning in unknown dynamic environment based on hybrid dsm model grid map. *Journal of Experimental & Theoretical Artificial Intelligence*, 23(1):5–22, 2011.

[175] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[176] Lim Chee Wang, Lim Ser Yong, and M. H. Ang Jr. Hybrid of global path planning and local navigation implemented on a mobile robot in indoor environment. In *IEEE Internatinal Symposium on Intelligent Control*, pages 821–826, 2002.

[177] Oliver Brock and Oussama Khatib. High-speed navigation using the global dynamic window approach. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 341–346, 1999.

[178] Dhanvin Mehta, Gonzalo Ferrer, and Edwin Olson. Autonomous navigation in dynamic social environments using multi-policy decision making. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1190–1197, 2016.

[179] Benjamin Riviere, Wolfgang Hoenig, Yisong Yue, and Soon-Jo Chung. Glas: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning. *arXiv preprint arXiv:2002.11807*, 2020.

[180] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[181] Xinlei Pan, Weiyao Wang, Xiaoshuai Zhang, Bo Li, Jinfeng Yi, and Dawn Song. How you act tells a lot: Privacy-leaking attack on deep reinforcement learning. In *International Conference on Autonomous Agents and MultiAgent Systems*, pages 368–376, 2019.

[182] Keisuke Okumura, Ryo Yonetani, Mai Nishimura, and Asako Kanezaki. Ctrms: Learning to construct cooperative timed roadmaps for multi-agent path planning in continuous spaces. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2022.

[183] Judea Pearl and Jin H Kim. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (4):392–399, 1982.

[184] Malik Ghallab and Dennis G Allard. A: An efficient near admissible heuristic search algorithm. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 789–791. Citeseer, 1983.

[185] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[186] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[187] Jinwoo Kim, Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. *Advances in Neural Information Processing Systems*, 35:14582–14595, 2022.

[188] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[189] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[190] Senthil Hariharan Arul, Adarsh Jagan Sathyamoorthy, Shivang Patel, Michael W. Otte, Huan Xu, Ming C. Lin, and Dinesh Manocha. Lswarm: Efficient collision avoidance for large swarms with coverage constraints in complex urban scenes. *IEEE Robotics Autom. Lett.*, 4(4):3940–3947, 2019.

[191] Bernard Chazelle. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM Journal on Computing*, 13(3):488–507, 1984.

[192] P Jiménez, Federico Thomas, and Carme Torras. Collision detection algorithms for motion planning. *Robot motion planning and control*, pages 305–343, 1998.

[193] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI Conference on Artificial Intelligence*, pages 2094–2100, 2016.

[194] Martín Abadi. Tensorflow: learning functions at scale. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, pages 1–1, 2016.

[195] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *International Conference on Learning Representations (ICLR)*, 2015.

[196] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[197] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[198] Alexandre Attia and Sharone Dayan. Global overview of imitation learning. *arXiv preprint arXiv:1801.06503*, 2018.

[199] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in Neural Information Processing Systems*, 29:4565–4573, 2016.

[200] Daewoo Kim, Sangwoo Moon, David Hostallero, Wan Ju Kang, Taeyoung Lee, Kyunghwan Son, and Yung Yi. Learning to schedule communication in multi-agent reinforcement learning. *International Conference on Learning Representations*, 2019.

[201] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *Advances in Artificial Life*, volume 929. Springer, 1995.

[202] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[203] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[204] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4243–4250. IEEE, 2018.

[205] Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 36(1):1–14, 2019.

[206] Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Deep drone acrobatics. In *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020.

[207] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[208] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Robotics: Science and Systems*, 2018.

[209] Rupert Mitchell, Jan Blumenkamp, and Amanda Prorok. Gaussian process based message filtering for robust multi-agent cooperation in the presence of adversarial communication. *CoRR*, abs/2012.00508, 2020.

[210] Jingwei Zhang, Lei Tai, Peng Yun, Yufeng Xiong, Ming Liu, Joschka Boedecker, and Wolfram Burgard. Vr-goggles for robots: Real-to-sim domain adaptation for visual control. *IEEE Robotics and Automation Letters*, 4(2):1148–1155, 2019.

[211] Rupert Mitchell, Jenny Fletcher, Jacopo Panerati, and Amanda Prorok. Multi-vehicle mixed reality reinforcement learning for autonomous multi-lane driving. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, page 1928–1930, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems.

[212] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[213] Baoqian Wang, Junfei Xie, and Nikolay Atanasov. Coding for distributed multi-agent reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 10625–10631. IEEE, 2021.

[214] Hoang M Le, Yisong Yue, Peter Carr, and Patrick Lucey. Coordinated multi-agent imitation learning. In *International Conference on Machine Learning (ICML)*, 2017.

[215] 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*, pages 6379–6390, 2017.

[216] Dave Coleman. Lee's o (n2 log n) visibility graph algorithm implementation and analysis, 2012.

[217] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

[218] Bo Li, Lionel Heng, Kevin Koser, and Marc Pollefeys. A multiple-camera system calibration toolbox using a feature descriptor-based calibration pattern. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1301–1307. IEEE, 2014.

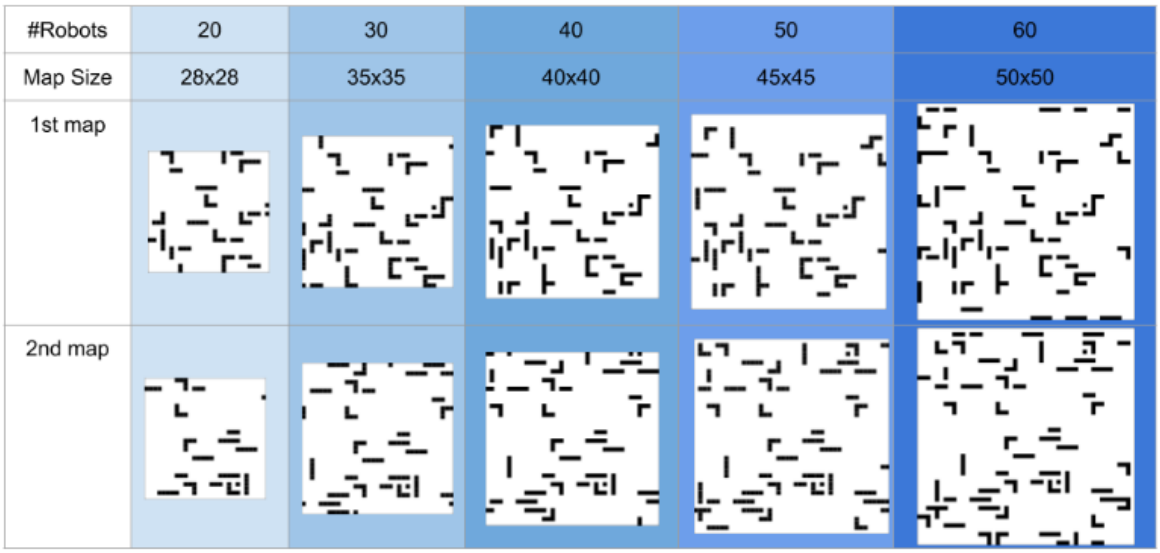
[219] Christopher Mei and Patrick Rives. Single view point omnidirectional camera calibration from planar grids. In *IEEE International Conference on Robotics and Automation*, pages 3945–3950. IEEE, 2007.

[220] Ajay Shankar, Sebastian G. Elbaum, and Carrick Detweiler. Freyja: A full multirotor system for agile & precise outdoor flights. In *IEEE International Conference on Robotics and Automation*, pages 217–223. IEEE, 2021.

Appendix A

Appendix for Chapter 3

]



(a) Random map

Fig. A.1 Two series maps from 28×28, 35×35, 40×40, 45×45 and 50×50 maps as exampples.

1) Dataset Generation Generate a set of large maps (50×50), then crop to the right size, where the effective density is consistent $\beta = \frac{n_{robots}+n_{obs}}{W \times H}$, where number of obstacles $n_{robots} = \rho \times W \times H$, is obstacle density in the map ($W \times H$), and n_{robots} is the number of robots, W, H

- 28×28 map with 20 robots –> 20 cases per map -> Choose 1000 cases for testing
- 35×35 map with 30 robots –> 20 cases per map -> Choose 1000 cases for testing

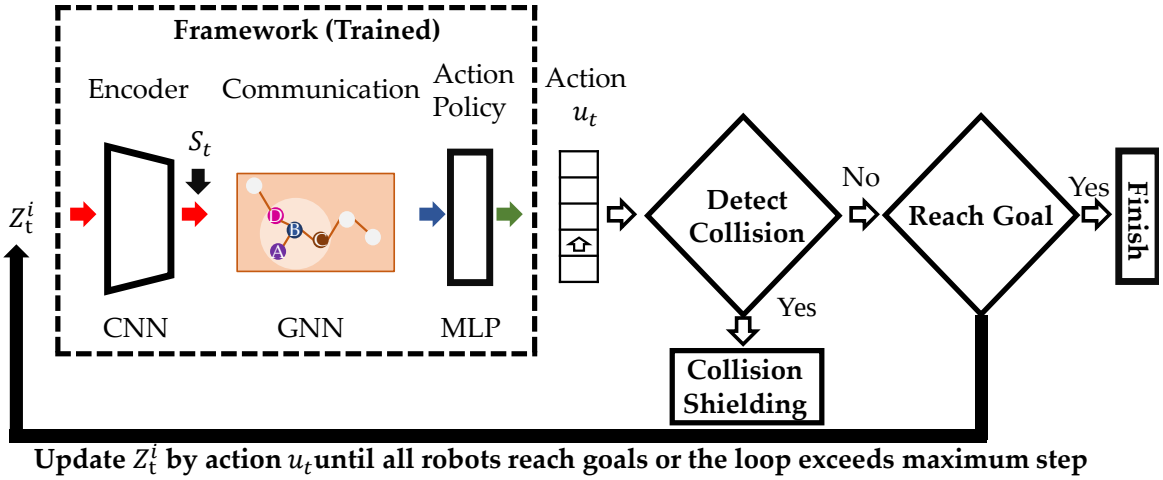


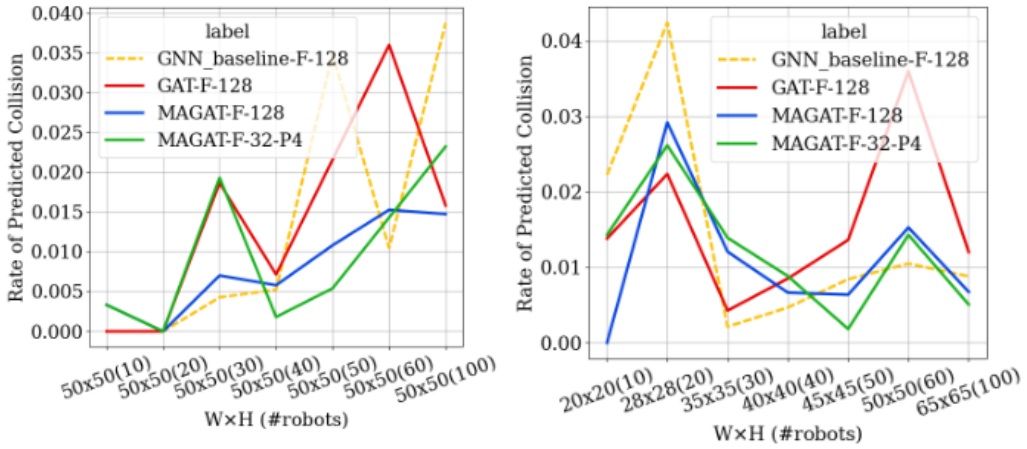
Fig. A.2 Illustration of the inference stage: for each robot, the input map $\mathbf{Z}_t^i$ is fed to the trained framework to predict the action; collisions are detected and prevented by collision shielding. The input map $\mathbf{Z}_t^i$ is continuously updated until the robot reaches its goal or exceeds the timeout T_{max}.

- 40×40 map with 40 robots –> 20 cases per map -> Choose 1000 cases for testing
- 45×45 map with 50 robots –> 20 cases per map -> Choose 1000 cases for testing
- 50×50 map with 60 robots –> 20 cases per map -> Choose 1000 cases for testing

2) Effectiveness of Collision Shielding Mechanism As illustrated in Fig. A.2: for each robot, the input map Z_t^i is fed to the trained framework to predict the action; collisions are detected and prevented by a protective mechanism called collision shielding. The input map Z_t^i is continuously updated until the robot reaches its goal or exceeds the timeout T_{Max}. Frankly, our model is not guaranteed that robots learn collision-free paths due to the limitation of imitation learning without explicit collision penalty. We require this additional mechanism to guarantee that no collisions take place. Collision shielding is implemented as follows:

- if the inferred action would result in a collision with another robot or obstacle, then that action is replaced by an idle action;
- if the inferred actions of two robots would result in an edge collision (having them swap positions), then those actions are replaced by idle actions. It is entirely possible that robots remain stuck in an idle state until the timeout is reached. When this happens, we count it as a failure case.

In order to verify that our proposed framework does not heavily rely on such protective mechanisms, we also recorded the average frequency of the predicted collisions of individual



(a) Increase Robot Density Set (b) Same Robot Density Set

Fig. A.3 Average frequency of the predicted collisions of individual robots ($\mu = \frac{1}{N_{\text{cases}}} \sum \frac{N^i_{\text{predictedcollision}}}{T^i_{\text{Steps}}}$) for two different testing set.

robots, which are prevented by the collision shielding mechanism introduced above. As shown in the above figure, given a communication bandwidth of 128, our MAGAT (Blue) can reduce the frequency of predicted collisions, compared with the GNN baseline (yellow) [19] and the original GAT [107] (red).

In all cases, our MAGAT (Blue) and the multi-head attention version (MAGAT-F-32-P4, green) can maintain the frequency of predicted collisions in low values. This video of the experiment on 50x50 with 50 robots also demonstrates how the robot makes collaborative decisions in crowded scenarios. Please also find this link[1] for the video demonstration.

[1]https://youtu.be/YKbhOlBqiOI?t=33

Appendix B

Appendix for Chapter 4

B.1 Appendix for Learning to Navigate using Visual Sensor Network

Parameters

1) Dataset In order to train the policy π, we imitate an expert path planner that operates in the continuous domain. The dataset is composed of random environment layouts and sensor placements. Each environment in the dataset has the same dimension $W \times H$ but is populated with a random number of obstacles of variable size. Obstacles are placed with a constraint on distance to other obstacles so that each obstacle C is placed not closer than D^C_{min} to any other obstacle. After placing obstacles, the target and sensors $S^2, \ldots, S^N$ are placed randomly in the environment, with the same distance constraint between sensors and obstacles, but with a separate distance constraint D^S_{min} between sensors and sensors. Lastly, the sensor node S^1 representing the robot R is placed with a distance of at least D^R_{min} to any obstacle or sensor. Eventually, we construct a visibility graph on an expanded map that accounts for the size of the robot, and we compute the shortest path from all sensors and the robot to the target using an any-angle variation of Lee's algorithm [216] computing a path $\mathscr{P}(q^R_0, q^G)$. If no valid path can be found for any of the sensors or robot, the map is discarded.

The dataset for all experiments, unless otherwise specified, contains environments of size $W \times H$ where $W = 9$ and $H = 12$, both units are number of boxes. Each box has has a size of 0.5 m $\times$ 0.3 m $\times$ 0.3 m, which is similar to the boxes used in the real-world setup. When placing obstacles, each obstacle is placed at least $D^C_{\text{min}} = 0.51$ m from any other obstacle, including the border. The robot and sensors are placed at a similar distance between each other and any obstacle, so that $D^R_{\text{min}} = D^C_{\text{min}}$. This is to ensure that the robot can always pass between any two sensors or obstacles. $N = 6$ Sensors are placed with a distance of

$D^S_{\text{min}} = 0.75$ between each other, to avoid a placement of multiple sensors too close to each other.

We compute 40000 environment layouts and then render the sensor views in Webots. We assume that sensor cameras are placed facing downwards with a fisheye lens, resulting in a 360° view of the environment. Due to OpenGL constraints, Webots does not support rendering of fisheye lenses. Instead, it supports a mode in which six camera images are automatically stitched together into a projection similar to a fisheye projection. Specifically, it projects the merged image to a square of configurable size, in our case 320 px × 320 px. We post-process this by projecting from a square to a circle and then projecting to a polar representation, using OpenCV, resulting in images of size 320 px × 120 px.

An environment sample, the corresponding environment in Webots and sensor samples can be seen in Fig. B.1.

2) Training

Policy. We use supervised learning to predict the pre-computed cost-to-go advantages. We predict $K = 8$ advantages (unless specified otherwise). The minibatch size is 16. The dataset contain 40000 samples, of which we use 80% for training, 29% for evaluation and 1% for testing. The test set is used for all evaluations and path visualization in this paper. The learning rate is 0.001 with an exponential decay of 0.98 with a seed of 1265. All experiments are trained for 200 training iterations, and the experiments with the best performance on the evaluation set is used to avoid overfitting. For evaluation, we use the L1 distance (mean absolute error) between predicted and ground truth cost-to-go advantages. We train with 16-bit floating-point precision.

Domain Adaptation. Training the translator is done with similar settings as the policy. We use the L2 distance (mean squared error) to evaluate the performance on the evaluation test. We use image augmentation to artificially increase the size of the dataset to avoid overfitting and increase robustness. We perform shifting (specifically rolling, as the image is a 360° view of the environment) of both the real image and the corresponding simulated image. We apply a random change of 50% to 150% in brightness, a random shift of ±9° in the hue color space, a random change of ±10% in contrast and ±5% in saturation as well as a random affine transformation with a rotation of ±3°, a translation and scale of ±3% and a shear of ±3° to the real image. Samples of this augmentation are also depicted in row 3 of Fig. B.3.

3) Neural Network Architecture The CNN is split in two halves for the sim-to-real domain adaptation process, where the first half creates an internal latent embedding as $\phi_{\text{PRE}} = y$ and the second half the communicated output features z so that $\phi(o) = \phi_{\text{POST}}(\phi_{\text{PRE}}(o)) = z$.

The feature extractor $\phi(\cdot)$ uses MobileNet v2 [162] to extract features z from the input omnidirectional image o. ϕ_{PRE} contains the first 14 residuals and ϕ_{POST} all others. The latent encoding z is a vector of size 64. We use PyTorch Geometric's GraphConv implementation for GNN layers with an output size of 256. The post-processing MLP consists of four layers with 128, 64, 64 and 64 neurons each and an output size of K for the number of cost-to-go advantages.

Real-world Experiments

The environment in Webots was co-designed with the real-world setup used for our experiments. We use cardboard paper boxes of size 0.5 m $\times$ 0.3 m $\times$ 0.3 m as obstacle and colored building boxes of size 0.1 m $\times$ 0.1 m $\times$ 0.1 m as sensor and target indicator. As sensor, we use a Raspberry Pi High Quality Camera equipped with a lens with a field of view (FOV) of 180°. The camera is connected to a Raspberry Pi 4 that runs Ubuntu 20.04 and ROS2 Foxy and performs image reading, performs projection from raw image to equirectangular [217] images, compression and streaming at a frame rate of 12 Hz. We use OpenCV and a custom-made camera calibration tool [218, 219] that allows us to automate most of the image calibration and cropping required to run the policy.

We use the same sensor contraption on the mobile robot. The Raspberry Pi is connected to the RoboMaster and performs low-level control using a custom made interface to the RoboMaster and the Freyja control suite [220].

1) Real World Dataset Collection We construct eight different environments and create a map using a motion capture system that we then transfer into our Webots simulation. A sample of a Twin Environment setup can be seen in Fig. B.2. We remote-control the robot manually while recording synchronized real and corresponding simulated images.

2) Interpreter To verify the functionality of our translator, which generates latent image encodings from real-world images, and to support debugging in the real-world deployment, we train a decoder $\phi^{-1}(\cdot)$ that maps the latent encodings z back to image observations o. Samples of reconstructions generated by the translator and the interpreter can be seen in Fig. B.3.

Table B.1 Analysis of the effect of number of cost-to-go advantages on performance.

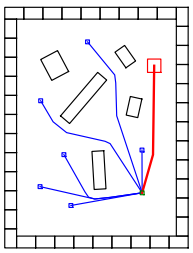
	LOS		NLOS	
K	Success	SPL	Success	SPL
8	1.000	0.945	0.837	0.711
16	1.000	0.967	0.913	0.801

Results

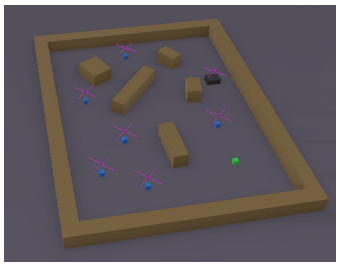
1) Trajectories In the following panels, we show all evaluation runs from the three real-world environments in Fig. B.5 (Environment A), Fig. B.6 (Environment B), and Fig. B.7 (Environment C).

We also report a broader selection of path evaluations in simulation in Fig. B.8.

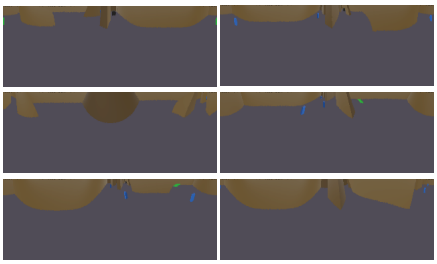
2) Number of cost-to-go advantages We compare the effect a varying number of cost-to-go advantages have on the policy. We compute $K = 8$ and $K = 16$ cost-to-go advantages and train two policies. The results can be seen in Tab. B.1 and show that the overall success rate increases and detour decreases with a larger K.



(a) Random map

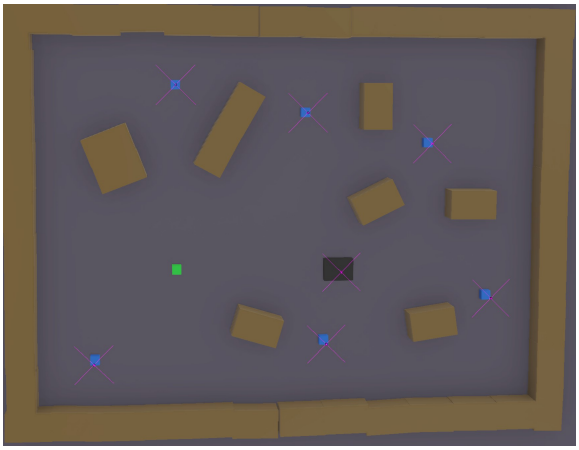


(b) Webots environment

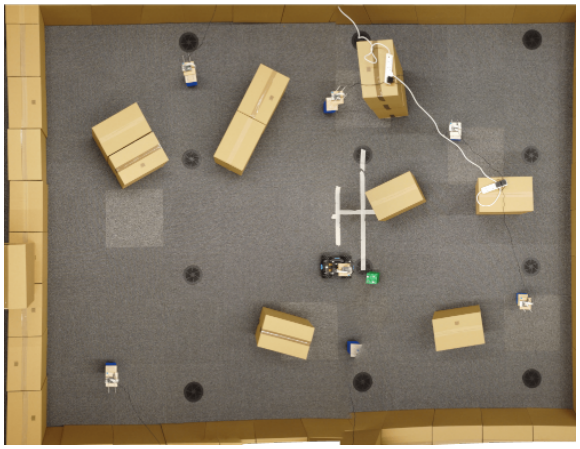


(c) Omnidirectional images from sensors

Fig. B.1 Generation of training data. (a) We first create random maps; in the illustration, blue points are sensor locations, the red square is the robot, green square is the target and corresponding lines indicate the shortest path from sensors and robots to the target. Black boxes indicate obstacles. (b) Random maps are rendered in 3D in the simulation environment, Webots. Sensors are blue, the target is green, the boxes brown and the robot black. (c) Sensors are equipped with omnidirectional cameras.



(a) Simulation in Webots.



(b) Corresponding real-world setup.

Fig. B.2 Sample of the Twin Environment setup. The simulated environment in Webots can be seen on the left, and the corresponding real-world environment on the right. Note the matching position and alignment of environment size and obstacles.

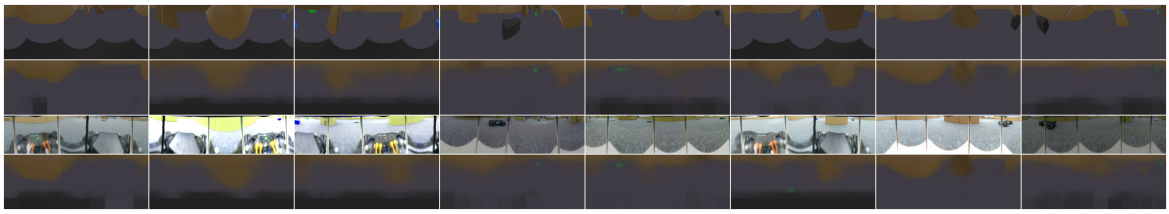


Fig. B.3 Samples of the interpreter used to convert latent image encodings into interpretable images. Columns: 8 independent samples. First row: Simulated image o^{sim}. Second row: The image in the first row is encoded as $z^{\text{sim}} = \phi(o^{\text{sim}})$ and reconstructed as $\phi^{-1}(z^{\text{sim}})$. Third row: Corresponding real-world image o^{real}. Fourth row: Reconstruction of the simulated image from the real image as $o^{\text{realtosim}} = \phi^{-1}(\phi_{\text{POST}}(\hat{\phi}(o^{\text{real}})))$.

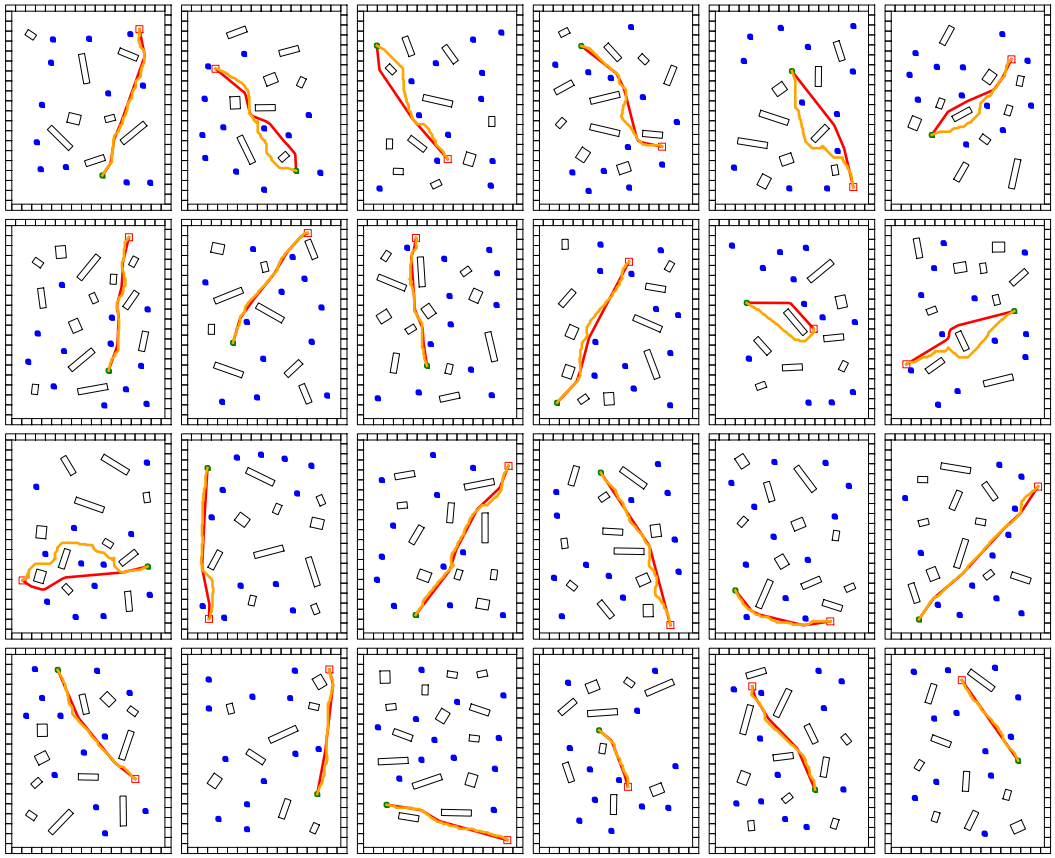


Fig. B.4 Generalizability to larger environments and larger number of agents $N = 13$ for GNN layers $L = 2$ and communication range $D_S = 3.5$ m (SPL 0.91). Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π.

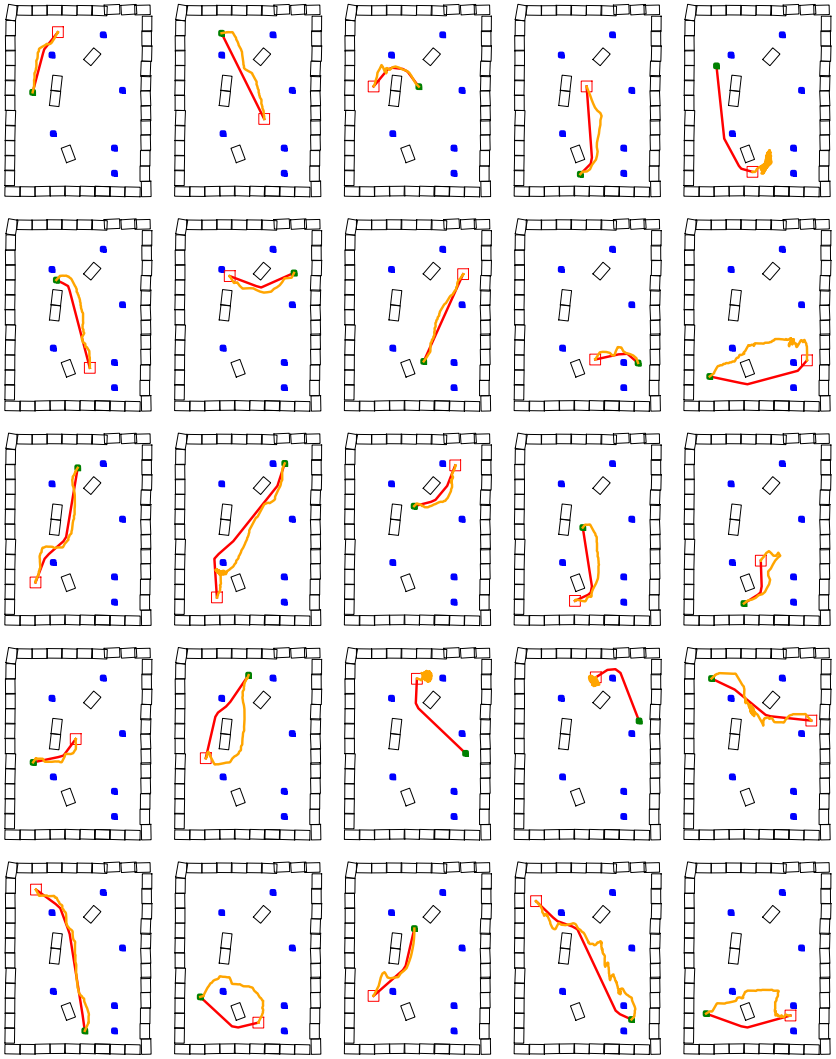


Fig. B.5 All real-world evaluations for Environment A. Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π.

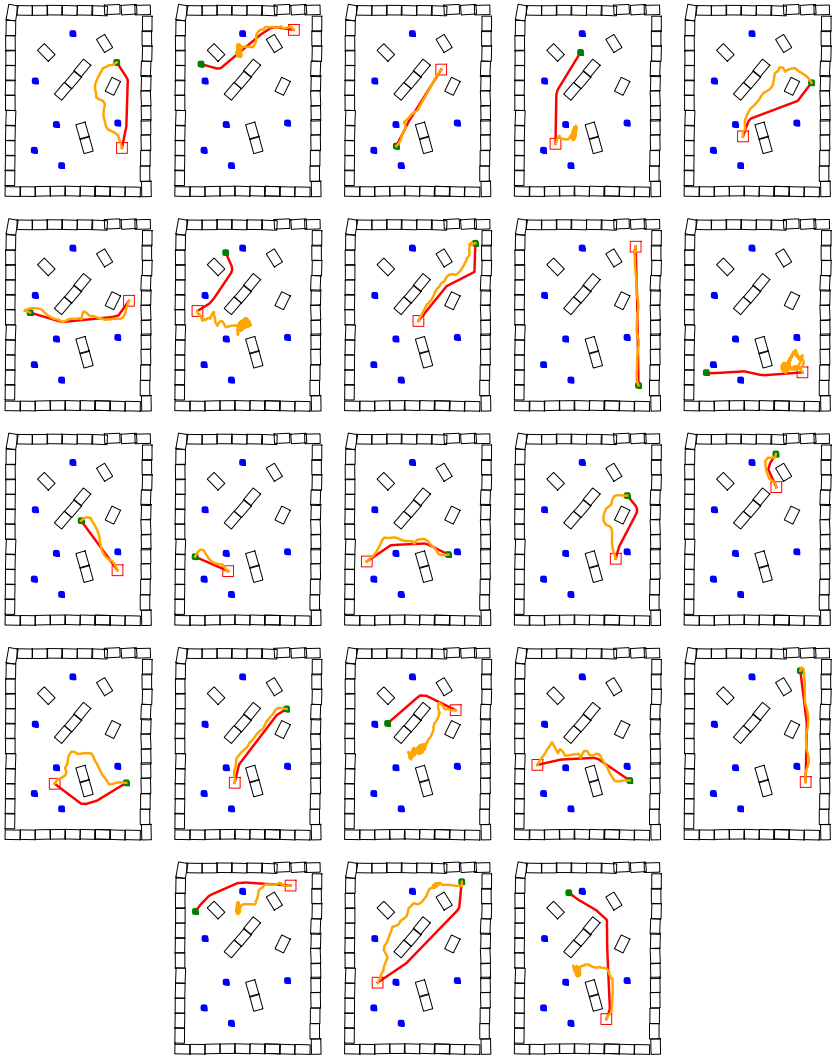


Fig. B.6 All real-world evaluations for Environment B. Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π.

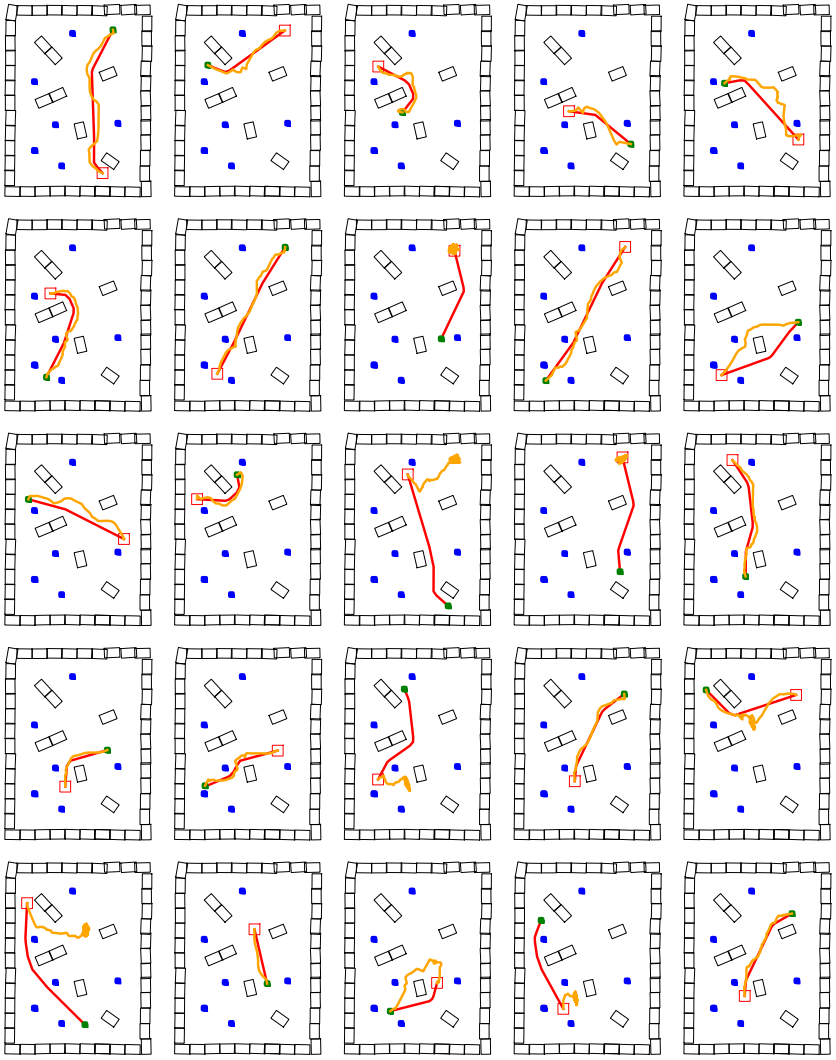


Fig. B.7 All real-world evaluations for Environment C. Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π.

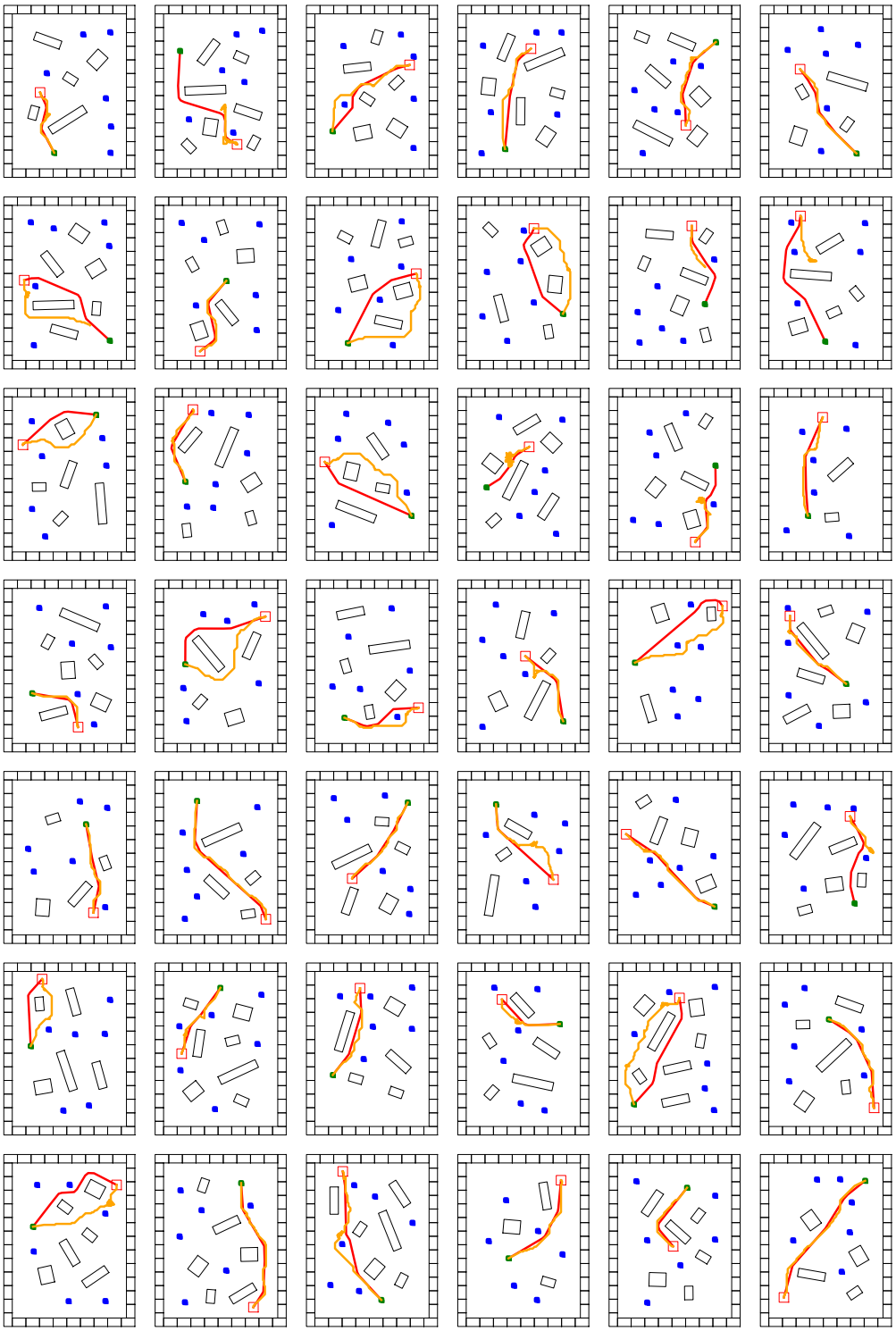


Fig. B.8 A selection of policy evaluations for NLOS configurations in simulation. Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π.

Appendix C

Appendix for Chapter 5

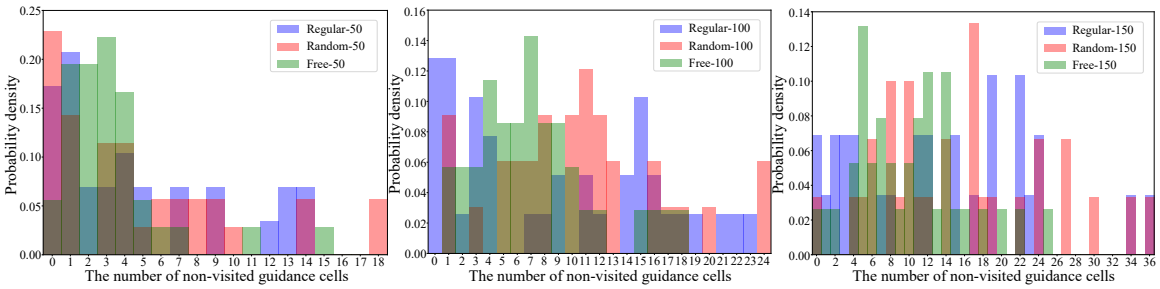
C.1 Appendix for Mobile Robot Path Planning in Dynamic Environments through Globally Guided Reinforcement Learning

Based on our reward function, there is no prior for the agent to strictly follow the global guidance. As long as the robot starts from the same location and chooses different paths (with the same number of movements) to reach the same guidance cell, the accumulated reward is the same. In order to validate this, we provide statistical results of our single robot experiments in Table. C.1 and Figure. C.1. More specifically, in Table. C.1, we first show the average number of eliminated global guidance cells that have not been traveled by the robot (non-visited guidance cells) in each environment map and its standard deviation across 100 tests. In Figure. C.1, we provide detailed results by using the histograms to show the data distribution. The results show that a large number of non-visited guidance cells exist in each experiment, which validates that our reward function provides dense rewards to ensure the convergence of the navigation task while simultaneously not requiring the robot to follow the global guidance strictly.

In addition, we also provide additional experiments to compare our reward function with a naive reward function, which encourages the robot to follow the global guidance strictly. The naive reward function is defined as follows: 1) A small negative reward r_1 each time the robot makes a movement; 2) An extra large negative reward r_2 if the robot conflicts with an obstacle, i.e., $R = r_1 + r_2 < 0$, where $r_2 < r_1 < 0$; 3) An extra large positive reward r_3 if the robot's next location is located on the global guidance, i.e., $R = r_1 + r_3 > 0$, where $r_3 > |r_1| > 0$; 4) An extra large negative reward $r_4 \times D_r$ if the robot's next location is not located on the global guidance, where D_r is computed by calculating the distance between

Table C.1 The number of eliminated global guidance cells which have not been travelled by the robot

	Mean	Standard Deviation
Regular-50	4.35	6.49
Regular-100	9.06	8.42
Regular-150	14.57	12.83
Random-50	5.08	5.01
Random-100	11.20	8.53
Random-150	15.29	11.43
Free-50	3.04	2.40
Free-100	7.12	4.45
Free-150	10.54	6.79



(a) The results with $||c_{goal} - c_{start}||_{L1} = 50$.

(b) The results with $||c_{goal} - c_{start}||_{L1} = 100$.

(c) The results with $||c_{goal} - c_{start}||_{L1} = 150$.

Fig. C.1 Histograms of the number of eliminated global guidance cells which have not been traveled by the robot across 100 tests in each environment map. $||c_{goal} - c_{start}||_{L1}$ represents the Manhattan distances between the start and goal cells, which are set to 50, 100, and 150.

the robot's next location with the nearest global guidance cell, i.e., $R = r_1 + D_r \times r_4 < 0$, where $r_4 < r_1 < 0$; 5) A small negative reward r_5 if the robot stays in its current location and does not move, i.e., $R = r_5 + D_r \times r_4 < 0$, where $r_5 < r_1 < 0$. In the experiments, we set $r_1 = -0.01$, $r_2 = -0.1$, $r_3 = 0.02$, $r_4 = -0.03$, and $r_5 = -0.015$. In experiments, we set a time-out for all the tests, within which time if the robot can not reach its goal, this test is defined as a failure case. In each test, the time-out value is set as double the Manhattan distance between the start cell and the goal cell of the robot. The results are shown in Table. C.2, in which the `Moving Cost` and `Detour Percentage` are calculated by only considering the successful cases. Since under the naive reward function, the robot is encouraged to follow the global guidance strictly, its performance is much worse than

those under our reward function. Strictly following the global guidance will introduce more detour steps and waiting time steps in the presence of motion conflicts, and may even cause deadlocks (which further lead to failure cases). In contrast, our reward function provides dense rewards while simultaneously not requiring the robot to follow the global guidance strictly. In this manner, we encourage the robot to explore all the potential solutions to reach the goal cell with the minimum number of steps.

Table C.2 Single robot path planning results: Comparing with naive reward based planner

	Success Rate		Moving Cost		Detour Percentage	
	Naive Reward	Our Reward	Naive Reward	Our Reward	Naive Reward	Our Reward
Regular-50	75%	**100%**	1.38(0.35)	**1.18(0.16)**	31.5(34)%	**15.2(15)**%
Regular-100	71%	**100%**	1.42(0.31)	**1.12(0.12)**	39.5(30)%	**10.7(12)**%
Regular-150	68%	**100%**	1.36(0.26)	**1.09(0.08)**	35.0(36)%	**8.2(8)**%
Random-50	80%	**100%**	1.36(0.28)	**1.21(0.13)**	30.1(35)%	**16.7(12)**%
Random-100	77%	**100%**	1.34(0.29)	**1.15(0.10)**	32.3(34)%	**13.0(10)**%
Random-150	73%	**100%**	1.40(0.28)	**1.11(0.08)**	39.4(40)%	**9.1(8)**%
Free-50	89%	**100%**	1.31(0.24)	**1.14(0.09)**	28.9(23)%	**12.3(9)**%
Free-100	83%	**100%**	1.34(0.25)	**1.11(0.07)**	33.6(25)%	**9.1(7)**%
Free-150	81%	**100%**	1.32(0.22)	**1.07(0.05)**	31.5(22)%	**6.5(5)**%

Values are listed as “mean (standard deviation)" across 100 instances. The lowest (best) values are highlighted.