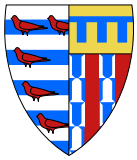
UNIVERSITY OF CAMBRIDGE

Learning-based cooperative perception and control for multi-robot systems

Jan Blumenkamp



Pembroke College

September 2024

This thesis is submitted for the degree of Doctor of Philosophy.

Personal Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the preface and specified in the text. It is not substantially the same as any work that has already been submitted, or, is being concurrently submitted, for any degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee.

Jan Blumenkamp

Words in text: 34437
Words outside text (captions, etc.): 3722
Words in appendix: 3895

Abstract

Multi-robot systems (MRS) are key to addressing complex real-world problems through the coordination of multiple autonomous agents [1] but are burdened by the curse of dimensionality, which compounds the difficulty of applying solutions to large-scale problem instances. The use of learning-based methods in multi-robot planning holds great promise as it enables us to offload the *online* computational burden of expensive centralised, yet optimal solvers, to an *offline* learning procedure [2].

Recent work has shown the promise of Graph neural networks (GNNs) to learn explicit communication strategies that enable complex multi-agent coordination [3, 4]. In the context of MRS, individual robots are modelled as nodes, the communication links between them as edges, and the internal state of each robot as graph signals. Sending messages over the communication links allows each robot in the graph to obtain an improved approximation of the global state, compared to relying on local information.

However, translating these policies from theoretical models to practical real-world MRS remains a critical challenge. Expert data is typically generated in simulations, yet policies trained in simulation often do not generalise to the real world. This lack of transferability is referred to as the *reality gap* [5], and transferring policies across this gap is a key challenge referred to as *sim-to-real transfer* [6].

While sim-to-real transfer in the single-robot domain typically deals with physical robot-world interaction, the multi-robot domain also concerns robot-robot interactions. Communication is vital to efficient multi-robot interaction [3, 4], but it is not yet clear how such communications are affected by the reality gap [5]. For instance, multi-robot coordination is typically trained synchronously, but when deploying these policies to the real world, decentralised communication is *asynchronous*. Furthermore, randomness, such as message dropouts and delays, is typically not considered during synchronous training.

There is a dearth of research that evaluates the *robustness of models* to such factors and their impact on the performance of policies. Decentralised mesh communication networks are required to operate MRS in the real world, which poses additional challenges to the sim-to-real transfer. This thesis presents a variety of different contributions and frameworks for deploying and testing multi-robot control and perception policies on decentralised real-world MRS.

The first part of this thesis focuses on applying learnable communications to multi-robot coordination. We are the first to train a cooperative multi-agent coordination policy with learnable communication using Reinforcement learning (RL) [7] through a differentiable communication channel and present simulation results for various multi-robot scenarios. To deploy these policies to the real world, we construct a fleet of nine agile ground-based robots, the Cambridge RoboMaster [8]. We use this platform to conduct a physical ablation of multi-agent deployment methods, from centralised to decentralised, and demonstrate the deployment of different coordination policies, showing the effects of sim-to-real transfer [9].

The second part focuses on learnable communications for multi-robot perception. Moving beyond coordination policies that rely on global state information from external localisation infrastructure, we explore training policies that utilise local sensor information from onboard monocular cameras. We train a GNNs-based visual navigation policy, guiding a mobile robot to a target without requiring calibration [10]. We then combine the insights on control and perception and introduce a cooperative, multi-robot visual spatial foundation model for decentralised perception and control for real-time real-world deployments [11]. This

model predicts relative poses and local spatial maps from monocular camera images and outperforms classical methods requiring overlapping camera views and predefined network infrastructures. We demonstrate the model's effectiveness by applying it to multi-robot formation control tasks, in which we show decentralised deployment in various indoor and outdoor scenes.

This thesis bridges the gap between theoretical learning and practical deployment in MRS. The developed methods pave the way for more efficient and adaptable MRS capable of addressing real-world challenges.

Acknowledgements

This thesis is the culmination of more than four years of hard work, countless experiments, unexpected challenges, and moments of discovery. While the research presented here bears my name, it would not have been possible without the support, guidance, and encouragement of numerous individuals who have contributed to both my work and personal growth throughout this journey.

First and foremost, I would like to express my gratitude to my advisor, Prof Amanda Prorok, who saw potential in me from the beginning. Your decision to select me for the MPhil preceding my PhD opened an important door in my academic journey. I am grateful for the trust you placed in me, the guidance that led to my first publication, and the opportunity to join your research group. Throughout my PhD, your continued mentorship and contributions to all my subsequent publications have been essential to their success. Our constructive meetings consistently provided clarity when progress seemed slow. Your constant encouragement to strive for excellence, together with guidance and reassurance during challenging times, has been invaluable. I also deeply appreciate the generous resources you provided for experiments, robots, conference travels and other research necessities that made this work possible.

I am grateful to my collaborators and co-authors: Steven Morad, Matteo Bettini, Ryan Kortvelesy, Ajay Shankar, Qingbiao Li, and Jennifer Gielis. Your insights, expertise, and collaborative spirit have significantly enhanced the quality of this research and made the journey intellectually stimulating. This was especially true during the challenges of the pandemic, when access to the department was severely limited. The work we produced together under such circumstances makes me even more proud of what we accomplished.

I would like to extend my appreciation to my PhD jury members, Prof Rafal Mantiuk and Prof Michael Otte, for their time and constructive feedback, that have significantly strengthened this thesis.

This research would not have been possible without the generous financial support of the Studienstiftung des deutschen Volkes. Their funding throughout my academic journey, from my undergraduate studies through my master's and doctoral research, has provided me with the freedom to pursue my research interests and the opportunity to focus on this work. I am grateful for their long-standing investment in my education and development as a researcher.

The camaraderie within the Prorok lab has made this journey considerably more enjoyable. Our trips to various locations all over the world that evolved into deep discussions about research and life have been highlights of my PhD experience. I've treasured not only these travels but also our board game evenings filled with competition and laughter, and our outings to various events around Cambridge that provided welcome breaks from research. I am also thankful to my friends at Pembroke College who played an important role in creating a rich social life during my time in Cambridge.

I would be remiss not to acknowledge those who sparked my interest in robotics long before my PhD. My gratitude goes to the robotics club during my high school years that enabled me to participate in international competitions which has opened many doors for me, and especially to Carsten Rohe, Armin Tischler and Dr. Rainer Ellerbrake at the Gymnasium Damme for nurturing my early interest in this field.

To my parents, Lena and Jürgen, and my sister Vanessa, I owe more than words can express. Euer Glaube an mich und eure unermüdliche Unterstützung, zusammen mit eurer grenzenlosen Liebe, waren stets mein stärkster Rückhalt. Ihr habt mich auf meinem akademischen Weg begleitet und meine Bildung immer an

erste Stelle gesetzt, oft auf Kosten eurer eigenen Wünsche und Bedürfnisse. Ihr habt jeden noch so kleinen Erfolg mit mir gefeiert und mich gelehrt, niemals aufzugeben. Die Werte, die ihr mir mitgegeben habt – Ausdauer, Fleiß und Aufrichtigkeit – haben mich durch diese Reise getragen. Ich bin euch unendlich dankbar für all die Opfer, die ihr gebracht habt, damit ich Chancen ergreifen konnte, die euch nicht zur Verfügung standen. Ich hoffe, dass euch dieser Abschluss mit ebenso viel Stolz erfüllt, wie ich Dankbarkeit für eure bedingungslose Unterstützung empfinde.

Finally, to Tina, your unwavering love, patience, and understanding have been my anchor throughout this challenging but rewarding process. The encouragement on difficult days, the celebrations of small victories, and your belief in me even when I doubted myself have meant more than words can express. As you embark on your own PhD journey, I cherish the opportunity to stand beside you with the same unconditional support and love you have shown me. Our shared experiences, both in academia and in life, have brought us closer with each passing day, and I look forward to all the chapters we have yet to write together.

Thank you,

Jan Blumenkamp

Contents

Abstract iii

Acknowledgements v

Contents vii

List of figures xi

List of tables xiv

List of symbols xv

List of supplementary material xvii

INTRODUCTION 1

1 Introduction 3
- 1.1 Research questions . . . 5
- 1.2 Thesis outline . . . 6
- 1.3 Contributions . . . 8

2 Background 11
- 2.1 Multi-agent systems . . . 11
  - 2.1.1 Benefits . . . 11
  - 2.1.2 Taxonomy . . . 12
- 2.2 Deep learning . . . 15
- 2.3 Learning from geometrical priors . . . 17
  - 2.3.1 Foundations . . . 17
  - 2.3.2 Graph neural networks . . . 19
  - 2.3.3 Convolutional neural networks . . . 21
- 2.4 Decision processes . . . 21
- 2.5 Reinforcement learning . . . 23
- 2.6 Deep reinforcement learning . . . 25
- 2.7 Multi-agent reinforcement learning . . . 27
- 2.8 Sim-to-real transfer . . . 28

3 Literature review 31
- 3.1 Multi-agent reinforcement learning . . . 31
- 3.2 Multi-robot systems . . . 32
- 3.3 Multi-robot research platforms . . . 34
- 3.4 Foundation models . . . 35
- 3.5 Sim-to-real transfer . . . 37
  - 3.5.1 Domain adaptation . . . 37
  - 3.5.2 Domain randomization . . . 37

3.5.3 Other methods . . . 38

Learning communication for multi-robot coordination **39**

4 Learning communication **41**
4.1 Contributions . . . 42
4.2 Approach . . . 42
4.2.1 Independent vanilla policy gradient . . . 42
4.2.2 Cooperative learning . . . 44
4.3 Experiments . . . 46
4.3.1 Environment setup . . . 47
4.3.2 Non-convex coverage . . . 47
4.3.3 Split coverage . . . 48
4.3.4 Path planning . . . 48
4.4 Results . . . 49
4.4.1 Metrics . . . 49
4.4.2 Baseline . . . 50
4.4.3 Performance evaluation . . . 50
4.4.4 Generalization . . . 52
4.4.5 Policy interpretation . . . 54
4.5 Conclusion . . . 58

5 Decentralised real-world control **61**
5.1 Contributions . . . 62
5.2 Approach . . . 62
5.2.1 Cambridge RoboMaster . . . 63
5.2.2 Deployment taxonomy . . . 64
5.2.3 ROS2 deployment . . . 65
5.3 Experiments . . . 67
5.3.1 Communication infrastructure . . . 68
5.3.2 Navigation in cluttered environments . . . 69
5.4 Results . . . 73
5.4.1 Networking . . . 73
5.4.2 Narrow constriction . . . 74
5.4.3 Scaling to eight robots in VMAS . . . 76
5.5 Conclusion . . . 78

Learning communication for cooperative perception **79**

6 Cooperative perception for visual navigation **81**
6.1 Contributions . . . 82
6.2 Approach . . . 82
6.2.1 Visual navigation using sensor networks . . . 83

6.2.2 Zero-shot real-world transfer . . . 88
6.3 Results . . . 90
6.3.1 Performance metrics . . . 91
6.3.2 Simulations . . . 91
6.3.3 Sim-to-real policy transfer . . . 93
6.4 Conclusions . . . 95

7 Visual spatial foundation models **97**
7.1 Contributions . . . 98
7.2 Approach . . . 98
7.2.1 Model . . . 99
7.2.2 Training . . . 102
7.3 Results . . . 105
7.3.1 Simulation . . . 105
7.3.2 Real-world . . . 108
7.3.3 Real-world pose control . . . 112
7.4 Conclusion . . . 115

Conclusion **117**

8 Conclusions **119**
8.1 Future work . . . 119
8.2 Closing remarks . . . 120

Appendix **123**

A Systems considerations **125**
A.1 Cambridge RoboMaster . . . 125
A.1.1 CAN communication . . . 125
A.1.2 Compute and components . . . 126
A.2 Fleet management . . . 127
A.2.1 Wireless communication . . . 127
A.2.2 Network . . . 128
A.2.3 Emergency stop . . . 128
A.2.4 On-robot user interface . . . 128
A.2.5 ROS2 drivers . . . 129
A.2.6 Decentralised deployment . . . 129
A.2.7 Multi-robot user interface . . . 129
A.3 Deployment . . . 130
A.3.1 Control stack . . . 130
A.3.2 Camera intrinsics . . . 130
A.3.3 Camera extrinsics . . . 130
A.3.4 Model deployment . . . 131

A.3.5 Ad-hoc WiFi 132

B Visual navigation **133**
B.1 Simulation 133
B.2 Real-world 136

C Visual spatial foundation models **141**
C.1 Simulation 141
C.2 Real-world 144
C.3 Real-world pose control 145

Bibliography **149**

Glossary **167**

List of figures

1.1 Applications of learning to optimisation problems . . . 3
1.2 Robot-world interactions . . . 4
1.3 Robot-robot interactions . . . 4
1.4 RQ1: Training cooperative multi-agent in sim . . . 5
1.5 RQ2: Transfer to robots in the laboratory . . . 6
1.6 RQ3: Transfer to the real world . . . 6
1.7 Training cooperative coverage . . . 6
1.8 Robots deployed in the laboratory . . . 7
1.9 Sensor-guided visual navigation . . . 7
1.10 Real-world foundation model deployment . . . 8

2.1 Single-agent system . . . 12
2.2 Multi-agent system . . . 12
2.3 Homogenous agents with no communication . . . 13
2.4 Heterogeneous non-communicating agents . . . 13
2.5 Homogenous communicating agents . . . 14
2.6 Heterogeneous communicating agents . . . 14
2.7 Cooperative perception in the warehouse . . . 15
2.8 Decision process taxonomy . . . 21
2.9 Markov decision process . . . 22
2.10 Sim-to-real delays . . . 28

3.1 Comparison of different multi-robot research platforms: cost vs speed . . . 34

4.1 Actor and critic neural network architectural overview . . . 44
4.2 Actor and critic neural network close-up . . . 44
4.3 Standard coverage environment . . . 47
4.4 Split coverage environment . . . 48
4.5 Path planning environment . . . 48
4.6 Ideal policy baseline with random baselines . . . 50
4.7 Generalization performance plot . . . 51
4.8 Communication hop performance increase . . . 52
4.9 Qualitative coverage for different environments . . . 53
4.10 Guided backpropagation in coverage task . . . 56
4.11 Global coverage reconstructed with learned interpreter . . . 58

5.1 RoboMaster close-up . . . 63
5.2 Real-world deployment configurations . . . 65
5.3 ROS2 architecture . . . 66
5.4 Multi-robot narrow constriction . . . 69

5.5 Narrow constriction guidance reward . . . 70
5.6 Model architecture overview . . . 72
5.7 Ad-hoc network performance evaluation . . . 74
5.8 Makespan and position distribution visualisations . . . 75
5.9 Four robots in roundabout scenario . . . 77
5.10 Eight robots facing each other . . . 78

6.1 Visual navigation setup with target, obstacles, a mobile robot, and movable wireless sensor . . . 82
6.2 Environment setup with obstacles, sensors, robot and target . . . 84
6.3 Visibility graph . . . 85
6.4 Model architecture overview . . . 87
6.5 Twin environment setup . . . 88
6.6 Physical sensor node and robot . . . 89
6.7 Sim-to-real transfer framework . . . 89
6.8 Policy evaluations trained on small and evaluated on large environment . . . 91
6.9 Communication range and number of sensors ablation . . . 92
6.10 Real-world policy evaluations . . . 93
6.11 Robustness against changing sensor layouts . . . 94

7.1 Three robots in pose formation task . . . 98
7.2 Overview of the model architecture . . . 99
7.3 Train and validation loss . . . 105
7.4 Simulation sample . . . 106
7.5 Simulation error distribution for position and rotation between robot pairs . . . 108
7.6 Dataset collection basestation . . . 109
7.7 Samples from the real-world dataset . . . 110
7.8 Real-world error distribution for position and rotation between robot pairs . . . 111
7.9 Real-world sample . . . 112
7.10 Multi-robot tracking performance evaluation . . . 113
7.11 Real-world deployment snapshots . . . 115

A.1 RoboMaster CAN protocol . . . 126
A.2 RoboMaster modifications . . . 127
A.3 Infrastructure overview . . . 127

B.1 Selection of simulation policy evaluations for NLOS configurations . . . 134
B.2 Generalizability to larger environments and larger number of agents . . . 135
B.3 Real-world evaluations for Environment A . . . 137
B.4 Real-world evaluations for Environment B . . . 138
B.5 Real-world evaluations for Environment C . . . 139

C.1 Simulation sample A . . . 141
C.2 Simulation sample B . . . 142
C.3 Simulation sample C . . . 142
C.4 Simulation sample D . . . 143

C.5 Simulation sample E 143
C.6 Real-world sample A and B 144
C.7 Real-world sample C and D 145
C.8 Additional real-world pose control snapshots 146
C.9 Statistics for the real-world dataset 147

List of tables

3.1 Comparison of different multi-robot research platforms: overview 34

4.1 Average returns for all cooperative agents over 100 episodes 51
4.2 Hyperparameter ablation summary 52
4.3 Performance $p_{1.0}$ for a all generalization trials. 53

5.1 Overview of performance metrics for all case study experiments. 74

6.1 Communication range ablation 91
6.2 Number of cost-to-go advantage ablation 93
6.3 Real-world policy evaluations 94

7.1 Simulation ablation study over the number of patches and size of features for each patch 107
7.2 BEV ablation 107
7.3 Real-world ablation study over the number of patches and size of features for each patch 109
7.4 First-principles baseline comparison 110
7.5 Median pose estimation error for each real-world scene 111
7.6 Median pose estimation error for different distance thresholds 111
7.7 Tracking error for both leaders for all three trajectories 114

A.1 RoboMaster cost breakdown 126
A.2 Model runtime evaluation 131

List of symbols

Notation	Description
$\mathbb{N}$	Set of natural numbers
$\mathbb{Z}$	Set of integers
$\mathbb{R}$	Set of real numbers
$\mathbb{B}$	Set of boolean values
$\mathcal{G}$	Graph (collection of nodes and edges)
$\mathcal{V}$	Set of vertices/nodes/agents/robots/sensors
$\mathcal{E}$	Set of edges in a graph
$\mathbf{A}$	Adjacency matrix of a graph
$\mathbf{P}$	Permutation matrix
R^i, S^i, C^i	Robot, agent, sensor, or camera i
N	Number of agents/robots
$\mathcal{N}_t^i$	Neighborhood of agent i excluding agent i
$\mathcal{N}_t^i \cup R^i$	Neighborhood of agent i including agent i
d	Communication range
K	Number of communication hops
$\mathcal{W}$	Environment space
$\mathcal{C}$	Obstacle space (subset of $\mathcal{W}$)
$\mathcal{A}$	Action space
$\mathcal{S}$	State space
$\mathcal{R}$	Reward space
$\mathcal{Z}$	Observation space
T	Number of time steps or horizon
γ	Discount factor
s_t	Global environment state at time t
a_t^i	Action for robot i at time t
$\mathbf{a}_t$	Joint action for all agents/robots at time t
z_t^i	Observation for robot i at time t
$\mathbf{z}_t$	Joint observation for all agents/robots at time t
r_t^i	Reward for agent/robot i at time t
$\mathbf{r}_t$	Joint reward for all agents/robots at time t
G_t	Joint return until time t
$Z(\cdot, \cdot)$	Observation function

Notation	Description
$V(\cdot)$	Value function
$V_\pi(\cdot)$	State-value function under policy π
$V_*(\cdot)$	Optimal state-value function
$Q(\cdot,\cdot)$	Action-value function
$Q_\pi(\cdot,\cdot)$	Action-value function under policy π
$Q_*(\cdot,\cdot)$	Optimal action-value function
$A(\cdot,\cdot)$	Advantage function
$\pi^i(\cdot)$	Control policy for agent i
$\boldsymbol{\pi}(\cdot)$	Joint control policy
$\pi_\theta(\cdot)$	Parameterized policy with parameters θ
$\mathscr{F}$	Function class of parameterized functions
Θ	Parameter space
$\boldsymbol{\theta}$	Parameters of function class (e.g., neural network weights)
$\mathbf{W}$	Weight matrix for a linear layer
$\mathbf{y}$	Output vector (or label vector)
$\mathbf{x}$	Input vector
$\mathbf{b}$	Bias vector
ν, ϕ, θ, ψ	Neural networks and their corresponding weights
$\mathbf{h}_t^i$	Embedding communicated for agent/robot i at time t
$\mathbf{h}_u$	Node representation/embedding for node u
$\mathbf{x}_v$	Node feature vector for node v
$\mathbf{e}_{uv}$	Edge features between nodes u and v
a_{uv}	Element of adjacency matrix representing edge between nodes u and v
$\mathscr{D}$	Dataset
$L(\cdot,\cdot)$	Loss function
$J(\cdot)$	Optimization objective
$\mathscr{X}$	Function space or signal space defined on domain Ω
$\mathfrak{G}$	Group (abstract algebraic structure with binary operation)
Ω	Domain on which signals are defined (e.g., grid, graph)
$\mathscr{C}$	Codomain representing space of values (e.g., pixel values, node features)
$\mathfrak{g}, \mathfrak{h}$	Elements of group $\mathfrak{G}$
$\rho(\mathfrak{g})$	Group representation (maps group element to invertible matrix)

List of supplementary material

- Github: *Adversarial Comms*

- Presentation: *CoRL 2020, Spotlight Talk 306: The Emergence of Adversarial Communication in Multi-Agent Reinforcement Learning*

- Supplementary video: *The Emergence of Adversarial Communication in Multi-Agent Reinforcement Learning*

- Github: *Multi-Agent RL passage formation for ICRA 2022*

- Supplementary video: *A Framework for Real-World Multi-Robot Systems Running Decentralized GNN-Based Policies*

- Github: *Sensor-Guided Visual Navigation*

- Supplementary video: *See What the Robot Can't See: Learning Cooperative Perception for Visual Navigation*

- Github: *Cambridge RoboMaster*

- Website: *The Cambridge RoboMaster*

- Supplementary video: *Emergency stop*

- Supplementary video: *VMAS eight robots*

- Supplementary video: *VMAS Square*

- Supplementary video: *VMAS Line*

- Website: *CoViS-Net: A Cooperative Visual Spatial Foundation Model for Multi-Robot Applications*

- Supplementary video: *CoViS-Net: A Cooperative Visual Spatial Foundation Model for Multi-Robot Applications*

Introduction

1 Introduction

1.1 Research questions 5
1.2 Thesis outline 6
1.3 Contributions 8

Multi-robot systems (MRS) are essential for solving complex problems through coordination among autonomous agents. Deploying multiple robots increases speed through parallelisation, improves robustness by avoiding single points of failure, and allows graceful degradation if individual robots fail. The system's modularity supports scalability and flexibility in adding agents, and it lowers development costs by promoting reusability and easier maintenance.

A major obstacle in deploying large-scale MRS is the curse of dimensionality. Learning-based methods in multi-robot planning have recently shown promise by shifting the computational burden from expensive *online* solvers to *offline* learning processes. Figure 1.1 illustrates how learning is applied to increase the scale of solvable problems, such as the Traveling salesperson problem (TSP) [12] and Vehicle routing problem (VRP) [13], or to increase the ability to deal with practical, partial information problems. Learning paradigms such as Reinforcement learning (RL) can be helpful in handling these problems.

[12]: Hudson et al. (2022), *Graph Neural Network Guided Local Search for the Traveling Salesperson Problem*

[13]: Nazari et al. (2018), *Reinforcement learning for solving the vehicle routing problem*

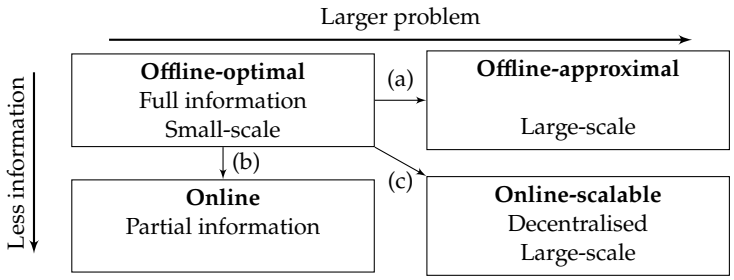


Figure 1.1: Applications of learning to optimisation problems: (a) embodies techniques for learning optimisation heuristics; (b) embodies techniques for learning to solve Partially observable markov decision processes (POMDPs); (c) is the emerging topic discussed here, embodying techniques for learning 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 (i.e., *'larger problem'*), while the ability to communicate and coordinate at scale decreases (i.e., *'less information'*). *Centralised* approaches produce optimal plans in the joint configuration space, but true optimality is NP-hard [14]. These approaches also struggle in environments with unreliable and degraded communications, thus requiring frequent replanning. *Decentralised* approaches reduce the computational overhead [15] and relax the dependence on centralised units to deal with

[14]: Yu et al. (2013), *Structure and intractability of optimal multi-robot path planning on graphs*

[15]: Desaraju et al. (2012), *Decentralized path planning for multi-agent teams with complex constraints*

challenged communications [16]. However, they primarily optimise local objectives and often fail to account for global optimisation.

[16]: Van den Berg et al. (2008), *Reciprocal velocity obstacles for real-time multi-agent navigation*

Effective communication is critical to decentralised control, but it remains to be seen *what information is crucial for agents to share to achieve task goals.* This question varies by problem, and optimal strategies are often unknown. Hand-engineered coordination methods require significant design effort and may only sometimes deliver the best performance.

MRS can be modelled as a graph, where robots are nodes, communication links are edges, and internal states are graph signals. Recent advances show promise in using Graph neural networks (GNNs) to learn communication strategies that allow agents to better estimate the global state by sharing key information and using this information for improved local decision-making [4]. Simulation-derived expert planners are used to generate offline data that can be used to train a model for fast online-inference with Imitation learning (IL) [3], or agents are instantiated in simulated worlds to learn from reward signals [7].

[4]: Gama et al. (2021), *Graph Neural Networks for Decentralized Controllers*

[3]: Li et al. (2020), *Graph neural networks for decentralized multi-robot path planning*

[7]: Blumenkamp et al. (2021), *The Emergence of Adversarial Communication in Multi-Agent Reinforcement Learning*

Even though simulations have become more realistic and easily accessible over recent years [17–19], it is often computationally infeasible to replicate all aspects of real-world physics in a simulation since the uncertainty and randomness of complex *robot-world* interactions are challenging to model (Figure 1.2). Models trained on simulated data often fail to generalise to real-world environments due to discrepancies in the modelling of physics, actuators or sensors, which is referred to as the *reality gap* [5].

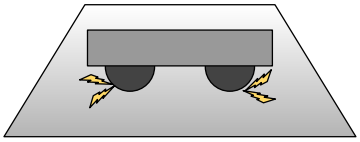


Figure 1.2: For any simulator, modelling robot-world interactions, such as friction, is a key challenge.

[17]: Todorov et al. (2012), *MuJoCo: A physics engine for model-based control*

[18]: Coumans et al. (2016), *PyBullet, a Python module for physics simulation for games, robotics and machine learning*

[19]: Savva et al. (2019), *Habitat: A Platform for Embodied AI Research*

[5]: Jakobi et al. (1995), *Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics*

For instance, the friction between objects or between the robot and the environment is highly complex and variable. Similarly, the mass distribution of components within a rigid body can differ between simulation and the real world. Actuators have backlash that is difficult to model. There are various delays from sensing over processing to transmission *within* a robotic system that will cause problems if not modelled properly. Lastly, visual sensors are notoriously hard to emulate in simulation, to an extent where simulated data is typically considered out-of-distribution of real-world data.

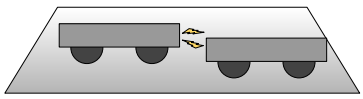


Figure 1.3: On top of robot-world interactions in single-robot systems, robot-robot interactions are important to model in simulators for MRS. This may include peer-to-peer communications.

On top of these challenges applicable to single-robot systems and MRS, the multi-robot domain also concerns *robot-robot* interactions (Figure 1.3). An example of this is collision avoidance between multiple mobile robots, or a swarm of drones flying in close proximity while turbulence affects the motions of other drones in the vicinity. We furthermore established that communication is important for efficient multi-robot interactions, but it is not clear how the reality gap, such as asynchronous communication, message dropouts, and delays, affects such communications. Multi-robot coordination is typically trained synchronously [3], but during the decentralised real-world deployment, peer-to-peer communication is typically *asynchronous*. Additional research evaluating the *robustness of*

models to such factors and their impact on the performance of policies is needed.

Furthermore, there are system-level considerations for deploying MRS. Decentralisation is key to scalable MRS. Therefore, decentralised mesh communication networks are required to operate MRS in the real world, which may pose additional challenges to the sim-to-real transfer due to interference and networking coordination conflicts. Operating and deploying control policies to fleets of robots poses additional logistics and systems challenges on its own, compared to deploying to a single robot.

This thesis addresses these challenges by presenting a variety of different contributions and frameworks for the deployment and testing of multi-robot control and perception policies on decentralised real-world MRS. We bridge the gap between theoretical learning and practical deployment in MRS and pave the way for more efficient and adaptable MRS capable of addressing real-world challenges.

1.1 Research questions

This thesis delves into the use of GNNs for training MRS for deployment in real-world scenarios. To guide our research, we pose the following research questions:

Question 1 *How can we train cooperative multi-agent control policies for dynamic, partial information environments?*

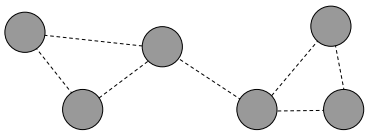


Figure 1.4: We answer the question of how we can train distributed coordination policies relying on communication, modelled as a graph, for MRS in simulation.

IL is only viable if expert data is available. Some problems have no known optimal solution, and obtaining optimal (or close to optimal) solutions is very expensive for many problems. RL promises to be a solution to these problems, but Multi-agent reinforcement learning (MARL) is neglected compared to single-agent RL. Existing MARL algorithms do not facilitate the learning of communication that would enable cooperative behaviours. How can we enable agents to learn cooperative behaviours while considering what information to communicate? What are the required theoretical foundations? There are also interpretability questions (e.g. what information is communicated, what kind of state representation the neural network learns), and how trained policies react to out-of-distribution deployments (e.g. to a smaller or larger number of agents than during training or in different environments). We illustrate this question in Figure 1.4.

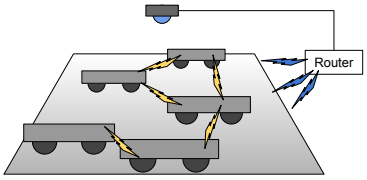


Figure 1.5: We proceed to answer the question of what considerations have to be met to transfer policies trained in simulation to laboratory settings. We assume that we have some fixed infrastructure, such as a tracking system, and find out what other deployment infrastructure is necessary. For instance, what is desirable in a fleet of robots, how can decentralised communication be realised, and how does the policy suffer from the reality gap?

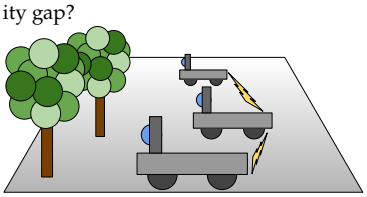


Figure 1.6: Lastly, we answer how to build and apply foundation models for MRS that allow deploying decentralised systems using on-board sensors to the real world.

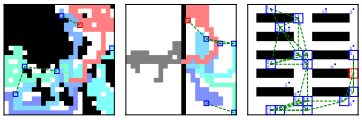


Figure 1.7: We train cooperative multi-agent coordination policies that rely on learnable communication in grid-based simulators.

Question 2 *How can we deploy multi-agent control policies trained in simulation to physical robots, and what impact does sim-to-real transfer have on the performance of MRS?*

Much work has been conducted on training agents in simulation, but the question of how to deploy them to real-world robots (or how to bridge the reality gap) remains open, particularly for MRS. In particular, it is unclear what the implications of asynchronous communication, message loss, and network delays are on the performance of the deployed policy on a system level and how future systems can be designed to be resilient to these challenges. We illustrate this question in Figure 1.5.

Question 3 *How can we train robust and reusable models, foundational for the use in decentralised MRS, acting entirely on onboard-sensing information?*

The expectation for any robot system is that it can operate fully autonomously in any given environment. For decentralised MRS, this includes relying on onboard sensing instead of external sensing (such as GNSS). Foundation models have become a standard in computer vision or natural language processing. However, it is unclear how such a model would look for MRS and what sensor modality is sensible. We illustrate this question in Figure 1.6.

1.2 Thesis outline

This thesis is structured around the three research questions. We split the thesis into two parts, which rely on learnable communication through differentiable communication channels. The first part focuses on multi-robot coordination, whereas the second focuses on cooperative perception. In the first part, we set the foundation for training cooperative multi-robot coordination policies in simulation. We then formulate and implement a framework to deploy these policies to custom-made robots. In the second part, we use similar, though more complex, model architectures as the first part and focus on monocular perception of the environment using onboard sensors. We first focus on the problem of sensor-guided visual navigation, where a robot is guided to a target through a distributed sensor network. Lastly, we create a foundation model that can be used in a wide range of multi-robot tasks.

We start with a background section in Chapter 2 to give the reader an overview of the foundations of this thesis. We continue with a literature review in Chapter 3 before we proceed with the main content.

In Chapter 4, we build the foundations for the first part of this thesis. We discuss Independent Vanilla Policy Gradient algorithms, how they work, and why they are inappropriate for learning cooperative behaviour based

on learnable communication. We introduce cooperative multi-agent reinforcement learning based on local rewards and provide theoretical insights and proof of optimality. We proceed to implement it based on Proximal policy optimization (PPO) in three different experiments, two of which focus on the Coverage path planning (CPP) problem and one on Multi-agent path planning (MAPF) (Figure 1.7). All environments use discrete actions in a simplified grid world. We perform an ablation study over various hyperparameters and study the policy's generalisation capabilities to up to 30 agents. Lastly, we interpret the trained policy using two white-box interpretability methods. We show that the GNN is building a global state understanding based on local information using a trained interpreter model, and we show what information is communicated in the coverage task using guided backpropagation [20]. This chapter is based on [7] and [21].

In Chapter 5, we focus on transferring trained policies on a decentralised MRS in the confines of a laboratory. To that end, we introduce a new multi-robot research platform, the Cambridge RoboMaster, as well as various infrastructure tools that aid real-world deployment. We then propose a taxonomy of different evaluation modes, from centralised to decentralised. We showcase the sim-to-real transfer on a constructed multi-robot coordination scenario, in which five robots start in formation, break formation to navigate through a narrow constriction, and then move back into formation as quickly as possible (shown in Figure 1.8). We present various real-world deployments and show the effect of the reality gap on multi-robot coordination policies trained in simulation. We close the chapter with an additional set of multi-agent navigation policies with up to eight robots in a centralised deployment mode. This chapter is based on [9] and [8].

We start the second part of the thesis, focusing more on cooperative perception, with Chapter 6. We investigate a single-robot target navigation scenario in which a robot has to localise and navigate to a target object. The environment is equipped with homogenous sensor nodes the robot can communicate with and use to navigate to the target object more quickly (refer Figure 1.9). We use IL with a custom dataset created in the Webots simulator to train a GNN. Notably, we only rely on visual information and require no position information, as the policy learns to infer the location of all sensors and the robot in the environment based on visual overlaps. We proceed with the zero-shot sim-to-real transfer based on a custom real-world dataset and construct custom physical sensor nodes for real-world deployment, which are still in the confines of the laboratory. We provide qualitative and quantitative simulation and real-world experiments, with an ablation study on the number of sensor nodes showing a significant improvement in performance with an increasing number of sensor nodes. We show that we can freely move sensors around and that the initial sensor placement is essential for the

[20]: Springenberg et al. (2015), *Striving for Simplicity: The All Convolutional Net*

[7]: Blumenkamp et al. (2021), *The Emergence of Adversarial Communication in Multi-Agent Reinforcement Learning*

[21]: Blumenkamp (2020), *Learning interpretable explicit communication for the multi-agent coverage problem*

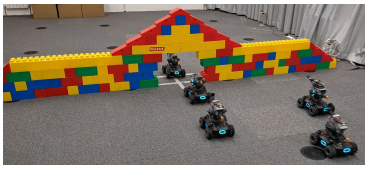


Figure 1.8: We deploy a set of five Cambridge RoboMasters in a real-world setup using GNNs and ad-hoc communication.

[9]: Blumenkamp et al. (2022), *A Framework for Real-World Multi-Robot Systems Running Decentralized GNN-Based Policies*

[8]: Blumenkamp et al. (2024), *The Cambridge RoboMaster: An Agile Multi-Robot Research Platform*

Figure 1.9: We demonstrate a sensor-guided visual navigation task in a laboratory environment.

[10]: Blumenkamp et al. (2023), *See What the Robot Can't See: Learning Cooperative Perception for Visual Navigation*

Figure 1.10: We demonstrate the decentralised real-world deployment of MRS outside the confines of the laboratory.

policy's successful operation. This chapter is based on [10].

Finally, in Chapter 7, we train a multi-robot foundation model that allows the transfer of trained coordination policies to a wide range of indoor and outdoor environments. We train a large multi-robot pose estimation and spatial aggregation model that predicts relative poses between robots and local bird's-eye-view maps only from onboard monocular visual sensors (refer Figure 1.10). This is achieved without creating a map and exclusively by learning spatial priors that help the model estimate relative poses between two camera perspectives, even if there is little to no visual overlap. We generate a custom dataset based on the Habitat simulator and propose a novel model architecture that allows pose estimation and spatial information aggregation. We then use uncertainty-aware training to predict poses and bird' s-eye-view maps and evaluate their performance on simulated and custom real-world datasets. We then transfer the model to our multi-robot platform on a leader-follower task. We use decentralised peer-to-peer communication and run the model onboard the robot without any external infrastructure. This chapter is based on [11].

[11]: Blumenkamp et al. (2024), *CoViS-Net: A Cooperative Visual Spatial Foundation Model for Multi-Robot Applications*

Lastly, we wrap up the thesis with the conclusion in Chapter 8, summarising our contributions and outlining future work.

1.3 Contributions

This thesis focuses on the foundational real-world deployment of learning-based multi-robot coordination strategies. Rather than improving on specific state-of-the-art methods, we provide novel mechanisms for learning and deploying multi-robot coordination policies. We summarise the main contributions as follows:

- We develop a new multi-agent learning model that integrates policies that share a differentiable communication channel, consisting of a decentralisable neural architecture that accommodates multiple distinct reward functions and a common learnable communication channel, and a reinforcement learning algorithm that elicits the emergence of cooperative communications from local rewards. [7]
- An omnidirectional indoor ground robot based on the DJI Robomaster S1, which is *affordable*, *agile*, *versatile*, and is easily *replicable* (we open source our entire stack, requiring minimal construction time), together with 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 centralised or decentralised manner. [8]

[7]: Blumenkamp et al. (2021), *The Emergence of Adversarial Communication in Multi-Agent Reinforcement Learning*

[8]: Blumenkamp et al. (2024), *The Cambridge RoboMaster: An Agile Multi-Robot Research Platform*

- We pioneer deploying trained cooperative multi-robot control policies decentralised to the real world. We propose several forms of deployment to quantify performance shifts ranging from centralised to decentralised with ad-hoc peer-to-peer networking. [9]
- We are first to show how low-cost visual sensor networks can help robots navigate to targets in unknown environments where neither the robot nor the sensors possess any absolute or relative positioning information. [10]
- We introduce a novel multi-robot foundation model that enables platform-agnostic, decentralised, real-time multi-robot pose estimation from monocular images, incorporating uncertainty awareness without a predefined map and uncertainty-aware BEV representations, facilitating spatial understanding in occluded areas. [11]

[9]: Blumenkamp et al. (2022), *A Framework for Real-World Multi-Robot Systems Running Decentralized GNN-Based Policies*

[10]: Blumenkamp et al. (2023), *See What the Robot Can't See: Learning Cooperative Perception for Visual Navigation*

[11]: Blumenkamp et al. (2024), *CoViS-Net: A Cooperative Visual Spatial Foundation Model for Multi-Robot Applications*

These contributions are drawn from the five primary conference papers related to this thesis:

- **Jan Blumenkamp**, and Amanda Prorok. "The emergence of adversarial communication in multi-agent reinforcement learning." In *Conference on Robot Learning (CoRL)*, pp. 1394-1414. PMLR, 2021.
- **Jan Blumenkamp**, Steven Morad, Jennifer Gielis, Qingbiao Li, and Amanda Prorok. "A framework for real-world multi-robot systems running decentralised GNN-based policies." In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 8772-8778. IEEE, 2022.
- **Jan Blumenkamp**, Qingbiao Li, Binyu Wang, Zhe Liu, and Amanda Prorok. "See What the Robot Can't See: Learning Cooperative Perception for Visual Navigation." In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7333-7340. IEEE, 2023.
- **Jan Blumenkamp**, Ajay Shankar, Matteo Bettini, Joshua Bird, and Amanda Prorok. "The Cambridge RoboMaster: An Agile Multi-Robot Research Platform." To appear in *International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2024.
- **Jan Blumenkamp**, Steven Morad, Jennifer Gielis, and Amanda Prorok. "CoViS-Net: A Cooperative Visual Spatial Foundation Model for Multi-Robot Applications." To appear in *Conference on Robot Learning (CoRL)*, 2024.

Code: Talk: Video:

Code: Video:

Code: Video:

Website: Code: Video:

Website: Code: Video:

The thesis builds upon work initiated for my master's thesis, from which some experimental results were included in Chapter 4:

- **Jan Blumenkamp**. "Learning interpretable explicit communication for the multi-agent coverage problem." Master's Thesis, *University of Cambridge*, 2020. [21]

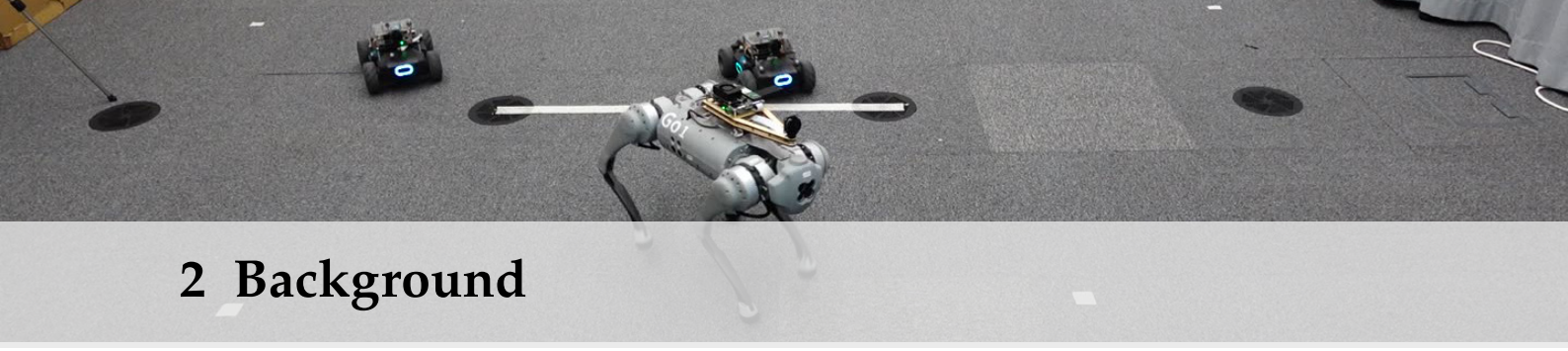
In addition to the five primary papers, the work on this thesis has contributed to five additional projects:

- Rupert Mitchell, **Jan Blumenkamp**, and Amanda Prorok. "Gaussian process based message filtering for robust multi-agent cooperation in the presence of adversarial communication." arXiv preprint arXiv:2012.00508 (2020).
- 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 *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1804–1808. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2022.
- Matteo Bettini, Ryan Kortvelesy, **Jan Blumenkamp**, and Amanda Prorok. "VMAS: A vectorized multi-agent simulator for collective robot learning." In *International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pp. 42-56. Cham: Springer Nature Switzerland, 2022.
- Henry Smith, Ajay Shankar, Jennifer Gielis, **Jan Blumenkamp**, and Amanda Prorok. "SO (2)-equivariant downwash models for close proximity flight." In *IEEE Robotics and Automation Letters (RA-L)*, 2023.
- Steven Morad, Ajay Shankar, **Jan Blumenkamp**, and Amanda Prorok. "Language-Conditioned Offline RL for Multi-Robot Navigation." Under review at *2025 International Conference on Robotics and Automation (ICRA)*, 2025.

Code: Video:

Website:

These projects explore related topics, such as detecting adversarial messages, simulation environments, applying GNN to drones, and combining Large language model (LLM) with MRS.



2 Background

In this chapter, we lay out the foundations for this thesis. We start by motivating Multi-agent systems (MAS) and MRS and reiterate a literature taxonomy into homogenous and heterogeneous and communicating and non-communicating agents in Section 2.1. We then introduce the foundations of deep learning in Section 2.2 and expand them with geometrical learning in Section 2.3. Based on this, we introduce Convolutional neural network (CNN) and GNN from a group-theoretical point of view. We then proceed to introduce decision processes in Section 2.4, which are foundational for RL introduced in Section 2.5, deep RL in Section 2.6, and MARL in Section 2.7. Lastly, we elaborate on the challenges of sim-to-real transfer in Section 2.8.

2.1 Multi-agent systems . . . 11
2.1.1 Benefits 11
2.1.2 Taxonomy 12
2.2 Deep learning 15
2.3 Learning from geometrical priors 17
2.3.1 Foundations 17
2.3.2 Graph neural networks . . 19
2.3.3 Convolutional neural networks 21
2.4 Decision processes 21
2.5 Reinforcement learning . 23
2.6 Deep reinforcement learning 25
2.7 Multi-agent reinforcement learning 27
2.8 Sim-to-real transfer 28

2.1 Multi-agent systems

The field of MAS generally refers to managing behaviour and interactions among collections of multiple dependent agents. It focuses on developing appropriate *coordination strategies* that enable these agents to operate effectively within their time and space constraints. In MRS, these agents are robots acting in the physical world. Each agent is an autonomous entity with specific *goals*, *actions*, and some degree of *domain knowledge*, operating within a dynamic environment. [1]

[1]: Stone et al. (2000), *Multiagent Systems: A Survey from a Machine Learning Perspective*

2.1.1 Benefits

MAS are particularly valuable and often necessary when different entities with distinct goals and local information must *coordinate* without centralised control. This approach is especially suited for complex, distributed tasks because it can parallelise operations robustly and scalable. In MRS, the use of multiple, specialised robots allows for *spatial distribution*, which enhances performance by enabling the system to observe the environment from multiple viewpoints, leading to a more comprehensive understanding. Additionally, redundancy achieves *robustness*, and the system's modularity facilitates *scalability*. Specifically, modularity allows adding or removing homogeneous agents and facilitates breaking down tasks into subtasks solved by heterogeneous agents. This applies similarly to the programming of a MRS, as identified subtasks can now be implemented in parallel by engineers, even for heterogeneous agents. [1,

pp. 4 sq.] Some applications are inherently multi-agent, such as interactions within complex systems like economies or supply chains. Other applications, while not inherently multi-agent, can significantly benefit from an increased number of agents by improving task speed through *parallelisation*, increasing robustness by avoiding single points of failure, and simplifying problem-solving through modularisation. [1, pp. 4 sq.]

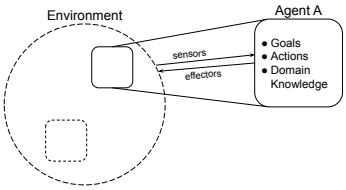


Figure 2.1: In a single-agent system, an agent models itself, the environment, and their interactions. Other agents are considered part of the environment. [1, pp. 7 sq.]

To define MAS further, we proceed to define single-agent systems. A single-agent system is a *centralised* system with a single agent making all decisions. It can have multiple entities (several actuators or robots), but if all are sent to a central process, there is only one agent. The system models itself, the environment, and the interactions between them. Even if there are other agents in the world, they are modelled as part of the environment without goals, actions, and domain knowledge. [1]

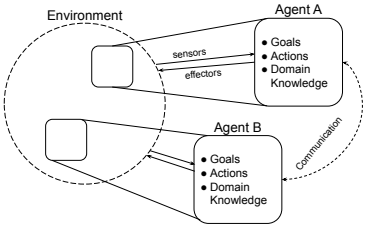


Figure 2.2: In an MAS, each agent is both part of the environment and modelled as a separate agent. Agents can be homogeneous (identical goals, actions and domain knowledge) or heterogeneous (different goals, actions and domain knowledge), and they can be communicating or non-communicating. [1, pp. 8 sq.]

In contrast, in an MAS, agents model each other's goals and actions, potentially while directly interacting with and communicating with each other. Most significantly, in an MAS, the environment dynamics can be determined by other agents. While in a single-agent system, the environment is static, and the agent is the only entity that can change it (only the agent has goals, actions and domain knowledge, whereas even if the environment changes dynamically, that dynamic would not be characterized as having goals, actions and domain knowledge), in a MAS, other agents interact with the environment and change it, adding to the uncertainty and complexity of the system. [1]

In summary, a MAS can provide the following benefits:

- *Necessity:* Some problems are inherently multi-agent, and it is often beneficial to observe the environment from multiple viewpoints.
- *Speedup and efficiency:* Adding more agents to the system can speed it up through parallelisation.
- *Robustness and reliability:* The system has no single point of failure and can degrade gracefully if agents fail.
- *Scalability and flexibility:* It is easy to add additional agents to the system if needed.
- *Lower development cost and re-usability:* It is easier to develop and maintain a modular system.

2.1.2 Taxonomy

MAS can be broadly classified along at least two axes: *homogeneous* vs *heterogeneous* agents and *communicating* vs *non-communicating* agents [1]. These classifications delineate the different strategies and challenges in multi-agent coordination, depending on whether the agents share the same structure and goals or have diverse objectives, and whether they

[1]: Stone et al. (2000), *Multiagent Systems: A Survey from a Machine Learning Perspective*

can communicate with one another or have to rely on indirect interactions within the environment.

Homogenous non-communicating agents are characterised by their similar structure. They have *identical* sensors, effectors, and domain knowledge but act on *different* information and thus have different outputs. If the information all agents are acting on were identical, the setup would be equivalent to a single-agent system [1].

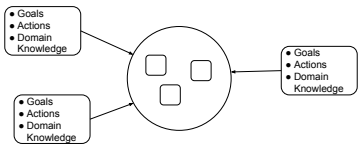


Figure 2.3: Homogenous agents with no communication: All agents have similar goals, actions and domain knowledge. [1, pp. 14 sq.]

The absence of a communication channel results in a unique set of challenges. The agents act solely on this *local* information and have no access to the other agents' observations or states. Therefore, homogenous non-communicating agents can generally not know other agent's actions due to limited information on internal state and sensory input, but they can make *predictions* of each other's actions given knowledge of each other's sensory input [1]. When global information is available, it can be beneficial to a MAS, for instance, to orchestrate global movements [22] while exploiting local properties of individual agents. On the other hand, acting on local rather than global information can be more effective, as local information can be more relevant to individual agent's goals, and global information can be overwhelming and irrelevant [23].

[22]: Shahrokhi et al. (2018), *Steering a Swarm of Particles Using Global Inputs and Swarm Statistics*

[23]: Roychowdhury et al. (1996), *Effects of local information on group behavior*
The term "ignorance is bliss" has been used to describe this phenomenon [23].

While explicit communication is not permitted, agents can communicate through environment interaction. This is referred to as *stigmergy* [24]. For example, agents can push each other and thereby impact each other's state, modify the environment or leave markers behind (active stigmergy) [25]. Lastly, agents can learn social conventions like traffic or navigation rules.

[24]: Holland (1996), *Multiagent systems: Lessons from social insects and collective robotics*
[25]: Goldman et al. (1994), *Emergent coordination through the use of cooperative state-changing rules*

Heterogeneous non-communicating agents differ from their homogeneous counterparts in their goals, actions and domain knowledge diversity. Each agent operating with *distinct* objectives and capabilities leads to complex system interactions. Depending on their goals, heterogeneous agents can behave *benevolently* or *competitively* towards each other. For example, agents with similar goals may *cooperate*, while agents with conflicting goals may *inhibit* each other [25].

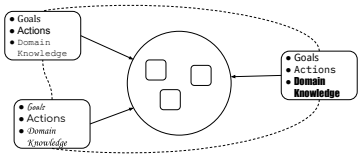


Figure 2.4: Heterogeneous non-communicating agents: Agents have different goals, actions and domain knowledge, shown with different fonts. [1, pp. 20 sq.]

The absence of communication exacerbates the challenges of coordination among heterogeneous agents. Unlike homogeneous systems, where agents might predict each other's internal state and thus actions since goals, actions and domain knowledge are shared, the diverse nature of heterogeneous agents makes it harder to anticipate behaviours and may require additional modelling of their goals and actions, but may not be possible depending on the scenario. This unpredictability can lead to a *dynamic environment* where agents must constantly adapt to their peers' changing behaviours.

In a learning context, the dynamic nature of environments inhabited by heterogeneous agents outside each other's understanding can lead to instability. As agents continuously adjust their strategies in response to others, this can trigger an "arms race" of adaptations, decreasing the system stability and the ability to attribute success or failure to specific agent behaviors.

This is known as the *credit assignment problem* [1], further elaborated on in Section 2.7.

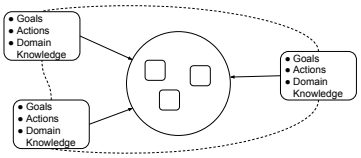


Figure 2.5: Homogenous communicating agents: Agents with similar goals, actions and domain knowledge and the ability to communicate. [1, pp. 28 sq.]

Homogeneous communicating agents share similar structures and capabilities and engage in explicit communication to enhance coordination. Communication significantly changes the system's dynamics, enabling agents to *share* critical information such as their goals, observations, and intentions that otherwise would have to be derived from local observation and domain knowledge. This is the class of agents we focus the work on in this thesis.

Communication among homogeneous agents can take various forms, including public message boards, broadcast messages to all peers, or point-to-point messages to specific peers. This interaction is usually considered *extra-environmental*, meaning it occurs outside the direct physical (tangible/material) context in which the agents operate. The communication content is crucial; for instance, sharing goals rather than states leads to more effective coordination among agents [26].

[26]: Balch et al. (1994), *Communication in reactive multiagent robotic systems*

However, the benefits of communication come with their own set of challenges. Bandwidth limitations can restrict the amount of information agents can share, necessitating efficient communication strategies where agents must learn *what* and *when* to communicate. Furthermore, delayed or outdated communications can become problematic, particularly in rapidly changing environments where real-time information is crucial for effective decision-making. Thus, agents must develop mechanisms to prioritise and filter the information they exchange to ensure their communications remain relevant and beneficial.

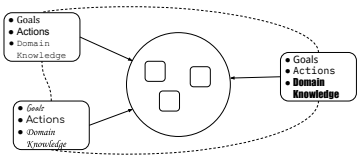


Figure 2.6: Heterogeneous communicating agents: Agents with different goals, actions and domain knowledge and the ability to communicate. [1, pp. 31 sq.]

Heterogeneous communicating agents combine the diversity of goals, actions, and domain knowledge found in heterogeneous agents with the benefits and complexities of explicit communication. The integration of communication in such a system can lead to interactions similar to those in single-agent systems.

In heterogeneous communicating systems, the decision to communicate (or not) can impact the system's overall efficiency and effectiveness. Agents need to develop or agree upon a shared protocol or language. This could be a predetermined vocabulary or evolve organically as agents learn to interact with each other. Effective communication is critical for negotiating shared goals, managing resources, and coordinating actions that consider each agent's varying objectives.

Resource management and negotiation become particularly significant in these systems, where agents must not only share information but also allocate limited resources to align with individual and collective goals. Moreover, commitment and decommitment become vital in managing interactions over time. Agents might commit to a specific course of action or collaboration for a period, enhancing the smoothness of the system's operations by reducing the need for constant renegotiation. However, they must also be able to reassess and adjust these commitments as new information becomes available or as the environment changes.

Finally, *collaborative localisation* is a practical application of heterogeneous communication systems, particularly in robotics. We provide an illustrated sample in Figure 2.7. Robots can collectively refine their understanding of their positions within an environment by sharing localisation data, leading to more accurate and efficient navigation and task execution. This example highlights the potential of heterogeneous communicating systems to combine the strengths of diversity and communication, achieving performance levels that would be unattainable by homogeneous or non-communicating counterparts.

While the majority of the work in this thesis uses homogenous hardware and neural networks, it will become clear that it is not easy to strictly separate these two classes in practice. We find that our homogenous control policies end up learning many of these properties, and to some extent learn to act heterogeneously in certain situations.

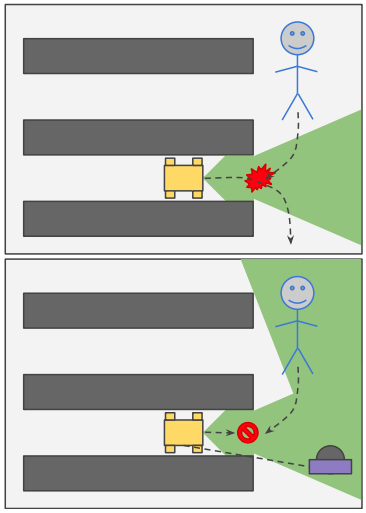


Figure 2.7: Top: Warehouse environment, shared between humans (blue) and robots (yellow). Humans are occluded by a shelf (grey), and robots cannot see (green), resulting in collision (red). Bottom: The robot communicates with strategically placed sensors (purple) to use this information for smarter decision-making, avoiding collisions before they occur.

2.2 Deep learning

In this section, we lay out the foundations for deep learning and the theory behind training deep neural networks. We define supervised learning, deep feedforward networks, and how to train them.

Supervised machine learning considers a set of N observations drawn from a dataset $\mathcal{D}$

$$\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=0}^{N}$$

i.i.d. from a data distribution P over $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are data and domain labels. Typically, $\mathcal{X}$ is a high-dimensional space like $\mathcal{X} = \mathbb{R}^d$, for example, a Euclidean space of large dimension d.

The labels $\mathbf{y}$ are generated by some unknown function f with $\mathbf{y}_i = f(\mathbf{x}_i)$. Our goal is to estimate f using the parameterised function class

$$\mathcal{F} = \{f_{\theta \in \Theta}\}$$

where the parameter space Θ is the set of all possible parameter values that the parameterised function class f_θ can take. Neural Networks are commonly used here, where θ corresponds to the neural network weights.

Deep feedforward networks are the basis for deep learning and usually consist of multiple composed functions representing an acyclic graph, referred to as the model. The more functions are composed, the deeper

the model, thus called deep learning. The way the layers are composed is called the model's architecture. The layers between the input and the output layers are called hidden layers. [27, pp. 164 sq.] A single linear layer can be defined as

$$\begin{aligned} \mathbf{y} &= \mathbf{W}\mathbf{x} + \mathbf{b} \\ \mathbf{x} \in \mathbb{R}^n, \mathbf{W} &\in \mathbb{R}^{m \times n} \\ \mathbf{b} \in \mathbb{R}^m, \mathbf{y} &\in \mathbb{R}^m \\ \hat{f}_n^{\boldsymbol{\theta}}(\mathbf{x}) &= \mathbf{y} \text{ with } \boldsymbol{\theta} = (\mathbf{W}, \mathbf{b}) \end{aligned}$$

In order to represent not only linear functions but also nonlinear functions, we introduce a nonlinear activation function σ, e.g. ReLU [28]:

[28]: Nair et al. (2010), *Rectified linear units improve restricted boltzmann machines*

$$\sigma(\mathbf{x}) = \max(0, \mathbf{x})$$

We can combine multiple instances of a single linear layer interleaved with activation functions to obtain a Multi layer perceptron (MLP) [29] with n layers.

[29]: Hornik (1991), *Approximation capabilities of multilayer feedforward networks*

$$\begin{aligned} \mathbf{y}_1 &= \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \\ &\dots \\ \mathbf{y}_n &= \sigma(\mathbf{W}_n\mathbf{y}_1 + \mathbf{b}_2) \\ \hat{f}_n^{\boldsymbol{\theta}}(\mathbf{x}) &= \mathbf{y}_n \text{ with } \boldsymbol{\theta} = (\mathbf{W}_1, \mathbf{b}_1, \dots, \mathbf{W}_n, \mathbf{b}_n) \end{aligned}$$

Modern deep-learning systems typically operate in the interpolating regime, where an estimated $\hat{f} \in \mathcal{F}$ satisfies $\hat{f}(\mathbf{x}_i) \forall i = 1, \dots, N$. The cost function J can be written as average over the training set as

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\tilde{P}} \left[L(\hat{f}(\mathbf{x}), \mathbf{y}) \right].$$

A common loss function is the squared loss

$$L(\mathbf{y}, \mathbf{y}') = \frac{1}{2} \|\mathbf{y} - \mathbf{y}'\|^2.$$

Due to the nonlinear nature of the model, the parameters $\boldsymbol{\theta}$ can only be found with iterative, gradient-based optimisers, which optimise a defined cost function to a low value. Applying the function f to $\mathbf{x}$, thus propagating $\mathbf{x}$ through the network, is called forward propagation, resulting in a scalar loss. Back propagation [30], on the other hand, allows information to flow backwards through the network, thus allowing the computation of the gradient. Other algorithms, such as stochastic gradient descent [31], perform the actual learning based on this gradient. Given the loss function L, the parameters $\boldsymbol{\theta}$, the inputs $\mathbf{x}$ and labels $\mathbf{y}$, we

[30]: Rumelhart et al. (1986), *Learning representations by back-propagating errors*

[31]: Kiefer et al. (1952), *Stochastic Estimation of the Maximum of a Regression Function*

want to find the parameters $\boldsymbol{\theta}$ that minimise the loss function as

$$\min_{\boldsymbol{\theta}} L(\hat{f}_n^{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}).$$

We compute the gradient of the loss function with respect to the parameters $\boldsymbol{\theta}$ as

$$\nabla_{\boldsymbol{\theta}} L = \frac{\delta L(\hat{f}_n^{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})}{\delta \boldsymbol{\theta}}$$

and iteratively update the parameters of the model with the gradient descent update rule until convergence with learning rate α

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(\hat{f}_n^{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}).$$

We typically use training sets $\mathcal{D}_{\text{train}}$, validation sets $\mathcal{D}_{\text{val}}$ and test sets $\mathcal{D}_{\text{test}}$ to evaluate the performance of the model.

2.3 Learning from geometrical priors

In the previous section, we explained how to train universal function approximators on data from high-dimensional spaces like $\mathcal{X} = \mathbb{R}^d$ with dimension d. While this approach theoretically scales to large d if the dataset is sufficiently large, there is practically an exponential requirement of data, referred to as the *curse of dimensionality*. [32]

[32]: Bronstein et al. (2021), *Geometric deep learning: Grids, groups, graphs, geodesics, and gauges*

2.3.1 Foundations

One way to solve this problem is by exploiting the underlying structure of many real-world problems. Specifically, we can use the concepts of symmetry and scale separation to exploit physically structured data. We will assume that our model operates on signals on some domain Ω:

Definition 2.3.1 *We define the function space $\mathcal{X}$ with a vector space structure for the set Ω and vector space $\mathcal{C}$ as*

$$\mathcal{X}(\Omega, \mathcal{C}) = \{x : \Omega \to \mathcal{C}\}$$

with addition and scalar multiplication defined as:

$$(\alpha x + \beta y)(u) = \alpha x(u) + \beta y(u) \forall u \in \Omega,$$

with $\alpha, \beta \in \mathbb{R}$. Given an inner product $\langle v, w \rangle_{\mathcal{C}}$ on $\mathcal{C}$, we can define an inner product on $\mathcal{X}(\Omega, \mathcal{C})$ as

$$\langle x, y \rangle = \int_{\Omega} \langle x(u), y(u) \rangle_{\mathcal{C}} du.$$

Ω is the domain on which signals x (e.g. images or graphs) are defined (e.g. a grid of pixels or a set of nodes). [32, pp. 11 sq.] The codomain $\mathcal{C}$ represents the space of values (e.g. pixel values or node features).

The domain Ω can be discrete, in which case the integral becomes a sum.

A group is a set $\mathfrak{G}$ with a binary operation $\circ : \mathfrak{G} \times \mathfrak{G} \to \mathfrak{G}$ called a composition (denoted as $\mathfrak{g} \circ \mathfrak{h} = \mathfrak{g}\mathfrak{h}$) that satisfies the axioms associativity, identity, inverse, and closure. [32, pp. 13 sq.] The group is defined as an abstract object specifying how objects compose, but not what the group elements are. We, therefore, introduce group actions and group representations.

One example for $\mathfrak{G}$ is the Euclidean group E(2), consisting of translations, rotations and reflections, which is the group of transformations of $\mathbb{R}^2$ [32, pp. 13 sq.]. This group can also be applied to the space of images.

A group action of $\mathfrak{G}$ on a set Ω is defined as mapping $(\mathfrak{g}, u) \to \mathfrak{g}.u$ that associates a group element $\mathfrak{g} \in \mathfrak{G}$ and point $u \in \Omega$ with some other point on Ω in a compatible way with the group operations.

$$(\mathfrak{g}.(\mathfrak{h}.x))(u) = ((\mathfrak{g}\mathfrak{h}).x)(u) \; \forall \, \mathfrak{g}, \mathfrak{h} \in \mathfrak{G}, u \in \Omega.$$

$\mathfrak{g}$ could be a translation or rotation. $\mathfrak{g}^{-1}$ is the corresponding inverse operation. $(\mathfrak{g}.x)(u)$ corresponds to the signal x after being transformed by $\mathfrak{g}$, evaluated at point u.

For the group $\mathfrak{G}$ acting on Ω, we obtain an action of $\mathfrak{G}$ on the space $\mathcal{X}(\Omega)$:

$$(\mathfrak{g}.x)(u) = x(\mathfrak{g}^{-1}u)$$

Linear group actions are referred to as group representations and can be described as maps $(\mathfrak{g}, x) \to \mathfrak{g}.x$ that are linear in x or as a map $\rho : \mathfrak{G} \to \mathbb{R}^n$ that assigns each group element $\mathfrak{g}$ to an invertible matrix $\rho(\mathfrak{g})$ [32, pp. 15 sq.]. While the dimensionality n can generally be arbitrary, in deep learning, it is typically the dimensionality of the group's feature space.

For example, the group of 2D translations acting on a space of images with n pixels.

Definition 2.3.2 *A n-dimensional real representation [32, pp. 15 sq.] of a group $\mathfrak{G}$ is a map $\rho : \mathfrak{G} \to \mathbb{R}^{n \times n}$, assigning to each $\mathfrak{g} \in \mathfrak{G}$ an invertible matrix $\rho(\mathfrak{g})$, and satisfying the condition*

$$\rho(\mathfrak{g}\mathfrak{h}) = \rho(\mathfrak{g})\rho(\mathfrak{h}) \; \forall \, \mathfrak{g}, \mathfrak{h} \in \mathfrak{G}$$

The action of $\mathfrak{G}$ on signals $x \in \mathcal{X}(\Omega)$ is defined as

$$\rho(\mathfrak{g})x(u) = x(\mathfrak{g}^{-1}u).$$

The symmetry of the domain Ω under the group $\mathfrak{G}$ can be exploited as a powerful inductive bias to reduce the complexity of the function class $\mathcal{F}$ [32, pp. 15 sq.]. Specifically, we can restrict the function class to be invariant or equivariant under the group action of $\mathfrak{G}$ on Ω:

Definition 2.3.3 *A function $f : \mathcal{X}(\Omega) \to \mathcal{Y}$ is $\mathfrak{G}$-invariant if*

$$f(\rho(\mathfrak{g})x) = f(x) \forall \mathfrak{g} \in \mathfrak{G}, x \in \mathcal{X}(\Omega),$$

i.e., its output is unaffected by the group action on the input [32, pp. 15 sq.].

One example of invariance is the shift-invariance typically used in image processing, where the output classification of the function is unaffected

by translations of the input image. The classification result should not be affected by the object's position in the image, which means that the function f is shift-invariant. CNN architectures that exploit this invariance have led to significant breakthroughs in image processing. [32, pp. 16 sq.]

Another important induction bias is equivariance [32, pp. 16 sq.], where the output of the function is affected by the group action on the input in the same way:

Definition 2.3.4 *A function* $f : \mathfrak{X}(\Omega) \to \mathfrak{X}(\Omega)$ *is* $\mathfrak{G}$*-equivariant if*

$$f(\rho(\mathfrak{g})x) = \rho(\mathfrak{g})f(x) \forall \mathfrak{g} \in \mathfrak{G},$$

i.e., group action on the input affects the output similarly. [32, pp. 16 sq.]

This is particularly useful for tasks where the output should be affected by the group action, such as in the case of a rotation-equivariant function that should output a rotated image when the input image is rotated. CNNs exploit this property in image segmentation task, for which a per-pixel classification has to be conducted.

2.3.2 Graph neural networks

A wide range of diverse real-world problems can be represented as graphs, making GNNs an essential class of model architectures. For instance, molecules and their relationships can be modelled as graphs, resulting in important applications for drug discovery problems [33]. Maps are represented as graphs, where intersections are nodes and street edges. Recently, Google Maps started using GNNs to improve ETA prediction [34]. Similarly, social networks can be modelled as graphs, thus making GNNs a possible model architecture for recommendation systems, e.g. for Pinterest [35] or Amazon [36].

[33]: Stokes et al. (2020), *A deep learning approach to antibiotic discovery*

[34]: Derrow-Pinion et al. (2021), *Eta prediction with graph neural networks in google maps*

[35]: Ying et al. (2018), *Graph convolutional neural networks for web-scale recommender systems*

[36]: Hao et al. (2020), *P-companion: A principled framework for diversified complementary product recommendation*

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a collection of nodes $\mathcal{V}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Each node has an s-dimensional node feature denoted as $\mathbf{x}_v \forall v \in \mathcal{V}$. Since the nodes in $\mathcal{V}$ are unordered, operations on the graph should not depend on the ordering of the nodes (the graph is permutation invariant). The neighbourhood set $\mathcal{N}_u$ of node u contains all edges connecting node u to v. For graphs, the domain $\Omega = \mathcal{G}$ and the space $\mathfrak{X}(\mathcal{G}, \mathbb{R}^d)$ is span by d-dimensional node-wise signals. The symmetry is given by the permutation group $\mathfrak{G} = \sum_n$ describing the possible permutations of orderings of node indices. [32, pp. 32 sq.]

To further explain the concept of invariance and equivariance on graph-structured data, we construct an $n \times d$-shaped matrix $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top$ consisting of stacked node features [32, pp. 32 sq.]. The permutation action

$\mathfrak{g} \in \sum_n$ results in reordering the rows of $\mathbf{X}$, which can be represented as $n \times n$ orthogonal permutation matrix $\rho(\mathfrak{g}) = \mathbf{P}$. We represent the graph connectivity as $n \times n$ adjacency matrix $\mathbf{A}$, defined as

$$a_{uv} = \begin{cases} 1 & (u, v) \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases}$$

A graph-wise function f is permutation invariant if

$$f(\mathbf{PX}, \mathbf{PAP}^\intercal) = f(\mathbf{X}, \mathbf{A})$$

and a node-wise function $\mathbf{F}$ is permutation equivariant if

$$\mathbf{F}(\mathbf{PX}, \mathbf{PAP}^\intercal) = \mathbf{PF}(\mathbf{X}, \mathbf{A})$$

for any permutation matrix $\mathbf{P}$ [32, pp. 33 sq.].

There are three spatial favours of GNNs: convolution, attention and message passing. In the convolutional flavour [37–39], features of the neighbourhood are directly aggregated as

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} c_{uv}\psi(\mathbf{x}_v)\right). \tag{2.1}$$

In the attentional flavour [40], the interactions are implicit

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} a(\mathbf{x}_u, \mathbf{x}_v)\psi(\mathbf{x}_v)\right), \tag{2.2}$$

where a is a learnable self-attention mechanism that implicitly learns some importance between two nodes. Lastly, in message-passing [41–43], arbitrary vectors, or messages, are computed across edges as

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v, \mathbf{e}_{uv})\right). \tag{2.3}$$

It is important to note that message-passing is the most powerful representation here, followed by attention and convolution.

Regardless of the flavour of the GNN, three possible operations can be performed on graphs. Firstly, node-wise operations perform some operation on each node output $\mathbf{h}_u$, e.g. a classification. Secondly, graph-wise operations are performed on the whole graph and require the graph to be reduced to some common representation $\bigoplus_{u \in \mathcal{V}} \mathbf{h}_u$. Lastly, link-wise operations are performed on edges $(u, v) \in \mathcal{E}$, e.g. by running some operation on the concatenation of node features $\mathbf{h}_u || \mathbf{h}_v$ [44].

[44]: Veličković (2023), *Everything is connected: Graph neural networks*

This work focuses on convolutional and message-passing GNNs, with node-wise or edge-wise operations.

2.3.3 Convolutional neural networks

Grids are a particular case of graphs with special adjacency but with a fixed order of nodes in this graph. Therefore, translation invariance is a stronger geometric prior here instead of permutation invariance. [32, pp. 36 sq.]

Convolution is essentially a way to measure the similarity between two functions. To compute the convolution of a signal x with some filter θ, we consider the inner product of x with a shifted version of θ with $\star : \mathscr{C}(\Omega) \times \mathscr{C}(\Omega) \to \mathscr{C}(\Omega)$
[32, pp. 60 sq.] as

$$(x \star \theta)(u) = \langle x, \theta_u \rangle$$

where $\theta_u(v) = \theta(u - v)$. We now substitute this into Definition 2.3.1 as

$$(x \star \theta)(u) = \int_{v \in \Omega} x(v)\theta(u - v)dv$$

From this, it is easy to derive the discrete convolution as

$$(x \star \theta)(u) = \sum_{v \in \Omega} x(v)\theta(u - v)$$

For an RGB image, $\Omega = \mathbb{Z}_n \times \mathbb{Z}_n$ (i.e. a two-dimensional $n \times n$ grid), the signal $x : \omega \to \mathbb{R}^3$ and f a function operating on $3n^2$ dimensional inputs.

2.4 Decision processes

We introduce four foundational decision processes for all concepts. We start with the MDP, which is the most specialised one, and its extension, the POMDP. We then introduce the Decentralized partially observable markov decision process (Dec-POMDP) and eventually the POMG, the most generalised decision process. Figure 2.8 shows a taxonomy of these decision processes.

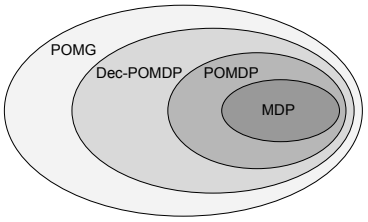


Figure 2.8: We show a taxonomy of decision processes, from Markov decision process (MDP) (most specialised) to Partially observable markov game (POMG) (most general).

An MDP [45] is a mathematical model that formally describes a sequential decision-making process in which an agent interacts with an environment over a series of discrete time steps. It is the foundation for RL and is defined as follows:

[45]: Howard (1970), *Dynamic programming and Markov processes*

Definition 2.4.1 *A Markov Decision Process (MDP) is a tuple* $(\mathcal{S}, \mathcal{A}, P, R)$ *where*

- $\mathcal{S}$ *is a finite set of states,*
- $\mathcal{A}$ *is a finite set of actions,*
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *is the state transition probability function, and*
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *is the reward function.*

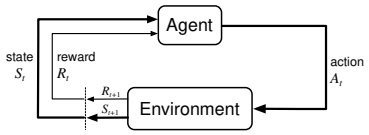


Figure 2.9: The agent-environment interaction in an MDP [46, pp. 38 sq.].

At each time step $t = 0, 1, 2, \ldots$, the agent interacts continuously with the environment by choosing an action $a_t \in \mathcal{A}(s_t)$ given the state $s_t \in \mathcal{S}$. For each interaction s_t and a_t, the agent receives a reward $R(s_t, a_t) = r_{t+1}$ from the reward function R and a new state s_{t+1} based on the state transition probability function P. This sequence results in a *trajectory* $s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \ldots$ [46, pp. 37 sq.].

MDPs model the agent with perfect state knowledge, but in many real-world and robotics applications, the agent has only partial knowledge of the environment. This discrepancy is addressed by POMDPs [47], which extend MDPs by introducing observations $\mathcal{Z}$ and observation function Z:

[47]: Åström (1965), *Optimal control of Markov processes with incomplete state information*

Definition 2.4.2 *A Partially observable markov decision process (POMDP) is a tuple* $(\mathcal{S}, \mathcal{A}, P, R, \mathcal{Z}, Z)$ *where*

- $\mathcal{Z}$ *is a finite set of observations,*
- $Z : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \to [0, 1]$ *is the observation probability function, and*
- *all other components are as in the MDP definition in Definition 2.4.1.*

Instead of observing the true state s_t, the agent receives an observation $z_t \in \mathcal{Z}$ based on the observation probability function Z, which the agent can use to infer the true state s_t. This reflects real-world robotics applications where sensors provide noisy partial information about the environment.

The previously defined MDPs only models single-agent systems, but in MAS, multiple agents interact with each other and the environment. As an extension, the Dec-POMDP was introduced [48]:

[48]: Shapley (1953), *Stochastic Games*

Definition 2.4.3 *A Decentralized partially observable markov decision process (Dec-POMDP) is a tuple* $(\mathcal{V}, \mathcal{S}, \mathcal{A}, P, R, \mathcal{Z}, Z)$ *where*

- $\mathcal{V} = \{1, \ldots, N\}$ *is a finite set of N agents,*
- $\mathcal{A} = \times_{i \in \mathcal{V}} \mathcal{A}_i$ *is a finite set of joint actions,*
- $\mathcal{Z} = \times_{i \in \mathcal{V}} \mathcal{Z}_i$ *is a set of joint observations,*
- *all other components are as in the POMDP definition in Definition 2.4.2.*

$\mathcal{A}_i$ is the set of actions and $\mathcal{Z}_i$ the set of observations available to agent $i \in \mathcal{V}$ (also referred to as robot R^i or sensor S^i), both of which can be different for each agent. The observation probability function Z defined

in Definition 2.4.2 uses the joint action and observation sets defined here. At each time step t, each agent executes action $a_t^i \in \mathcal{A}_i$. The agents form a joint action $\mathbf{a}_t \in \mathcal{A}$. The state transition probability function conditions the next state on the current state and the joint action as $P(s_{t+1}|s_t, \mathbf{a}_t)$. We consider a partially observable setting, in which each agent i draws observations $z_t^i \in \mathcal{Z}^i$ according to the observation function $Z(s_t, \mathbf{a}_t)$ forming a joint observation $\mathbf{z}_t \in \mathcal{Z}$.

While the Dec-POMDP models most aspects of a real-world MAS, it assumes a single reward signal for the whole team, which assumes that the credit assignment problem can be solved [49]. To alleviate this and allow local rewards, the POMG was introduced [47]:

[49]: Oliehoek et al. (2016), *A Concise Introduction to Decentralized POMDPs*
[47]: Åström (1965), *Optimal control of Markov processes with incomplete state information*

Definition 2.4.4 *A Partially observable markov game (POMG) is a tuple* $(\mathcal{V}, \mathcal{S}, \mathcal{A}, P, \mathcal{R}, \mathcal{Z}, Z)$ *where*

- $\mathcal{R} = \times_{i \in \mathcal{V}} R^i$ *is a finite set of reward functions,*
- *all other components are as in the Dec-POMDP definition in Definition 2.4.3.*

Individual agents now observe a local reward r_t^i drawn from $R^i(s_t, a_t^i) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (i.e., it is not a global reward given to the whole team).

2.5 Reinforcement learning

This chapter introduces the core concepts and ideas of RL. We introduce deep RL and the RL algorithms used in this thesis. RL problems generally consist of an *agent*, an *environment* the agent acts in, a deterministic or stochastic *policy* that determines the action from an observation, a *reward* signal for desirable short-term actions and a *value function* for computing the expected long-term cumulative reward or *return*. In the following, we introduce these concepts in more detail.

The reward signal is the primary means to adapt the policy to maximise the cumulative long-term reward. While rewards can be considered a primary sense that specifies whether an immediate action was good or bad, values are a secondary sense that considers the following observations and actions. Rewards and values are codependent: values cannot be predicted without rewards and have to be predicted to maximise the return [46, pp. 59 sq.].

The goal in RL is to maximise the long-term reward, or expected return G_t, which can be expressed as

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T$$

for the final time step T. This is only possible for episodic tasks, where agent-environment interactions can be split into *episodes* representing repeated interactions. Each episode is independent from the previous episode but may have varying lengths T. Some tasks, such as control tasks, may have no end (an episodic duration of $T = \infty$). This leads to an infinite expected return for the previous definition, which makes it impossible to assign values to sequences of actions. This problem can be solved by discounted rewards [46, pp. 60 sq.].

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \end{aligned}$$

The parameter $0 \leq \gamma \leq 1$ is referred to as *discount factor* and represents the value of future rewards: If γ is chosen to be zero, only the immediate reward r_{t+1} would be maximised and the closer γ converges to 1, the more heavily long-term rewards are considered [46, pp. 60 sq.].

Value functions estimate how good it is to be in a certain state or to perform a certain action in a certain state in terms of expected future return with respect to a policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. The policy maps states to probabilities of choosing a certain action. The expected return when starting in a state s and following the policy π from there on is the value $V_\pi(s)$. This is also referred to as *state-value* for policy π [46, pp. 70 sq.].

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[G_t \mid s_t = s] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right] \forall s \in \mathcal{S} \end{aligned}$$

The *action-value* function for policy π estimates the expected return when starting in a state s, choosing the action a and following the policy π.

$$\begin{aligned} Q_\pi(s, a) &= \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a \right] \forall s \in \mathcal{S}, a \in \mathcal{A} \end{aligned}$$

While it is possible to estimate the value function from experience by keeping track of all states and returns encountered from that state and averaging them, this quickly becomes computationally infeasible. Instead, a parameterised function with fewer parameters than states must be maintained and adjusted to the observed returns [46, pp. 70 sq.].

An essential property of the value function is that it satisfies a recursive relationship to the value of its successor state:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[G_t \mid s_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V_\pi(s')\right] \forall s \in \mathcal{S} \end{aligned}$$

This equation is referred to as the *Bellman equation* for V_π for which it is the unique solution. It averages all possible combinations of actions a for the current state s, possible subsequent states s', and rewards r weighed by the corresponding probability p.

The goal in RL is to find the *optimal policies* π_*, which are guaranteed to exist and have a greater expected return than all other policies π' for all states. These optimal policies have optimal state-value functions V_* and action-value functions Q_*, which can be expressed through V_* (since the action-value function expresses the expected reward after if the given action a is executed in the given state s and following the policy π) [46, pp. 71 sq.].

$$\begin{aligned} V_*(s) &= \max_\pi V_\pi(s) \forall s \in \mathcal{S} \\ Q_*(s, a) &= \max_\pi Q_\pi(s, a) \forall s \in \mathcal{S}, a \in \mathcal{A} \\ &= \mathbb{E}[r_{t+1} + \gamma V_*(s_{t+1}) \mid s_t = s, a_t = a] \end{aligned}$$

Finding optimal policies and value functions requires extensive computation and memory cost. Even though optimal policies are guaranteed to exist for fully observable MDPs, it is only possible to find approximations in the form of parametrised functions in practice. The key idea in RL is to use value functions to structure the search for policies [46, pp. 75 sq.]. While this is traditionally done with table-based approaches, dynamic programming, Monte-Carlo methods or Temporal-Difference Learning, deep learning is naturally able to approximate these methods and functions [46, pp. 82 sq.].

2.6 Deep reinforcement learning

The first deep learning approach to RL was proposed as Deep Q-learning (DQN) [50]. The action-value function $Q_*(s, a)$ is approximated with a Neural network (NN) with parameters θ so that $Q_\theta(s, a) \approx Q_*(s, a)$, thus extending traditional Q-Learning. This approximator can be trained by minimising the loss $L(\theta)$:

[50]: Mnih et al. (2013), *Playing Atari with Deep Reinforcement Learning*

$$
\begin{aligned}
L(\theta) &= \mathbb{E}_{s,a\sim p(\cdot)}\left[(y_i - Q_\theta(s,a))^2\right] \\
y &= \mathbb{E}_{s'\sim\epsilon}\left[r + \gamma \max_{a'} Q_{\theta_{\text{old}}}(s',a')|s,a\right]
\end{aligned}
\tag{2.4}
$$

where y is the optimization target at the current iteration, $p(s,a)$ is a distribution over states and actions and ϵ is the simulation environment (in case of DQN an Atari 2600 emulator). This loss is differentiated and optimised with stochastic gradient descent. This approach is *model-free*, meaning it can understand the environment from observations and off-policy because it uses the behavioural distribution instead of a previous policy to learn. Sampling from a *replay buffer* is advantageous because it allows reusing older samples (resulting in a higher data efficiency), and it removes correlation between samples and thus the variance in the policy updates and generally making the learning more stable through averaging over these previous samples [51].

[51]: Mnih et al. (2016), *Asynchronous Methods for Deep Reinforcement Learning*

A major disadvantage of DQN is that it requires a discrete action space, while many real-world problems require a continuous action space. Besides Q-Learning-based algorithms, *policy gradient-based algorithms* are the second primary class of model-free algorithms. The general idea of policy gradient algorithms is to estimate the gradient g of a stochastic policy π_θ through the *advantage function* A_t and maximise it through stochastic gradient ascent.

$$
\begin{aligned}
g &= \mathbb{E}_t\left[\nabla_\theta \, log \, \pi_\theta(a_t|s_t) A_t\right] \\
A_t &= Q_\pi(s_t,a_t) - V_\pi(s_t), s_t \in \mathcal{S}, a_t \in \mathcal{A}, t \geq 0
\end{aligned}
\tag{2.5}
$$

The advantage function requires a value function estimate, commonly predicted by a dedicated NN and trained in parallel with the action predictor. Such architectures are referred to as *actor-critic* [52, 53].

[52]: Konda et al. (2000), *Actor-Critic Algorithms*

[53]: Peters et al. (2008), *Natural Actor-Critic*

Empirically optimising this gradient leads to large policy updates that result in instability. Trust region policy optimization (TRPO) [54] attempts to solve this problem by constraining the policy update of an objective function (the fraction of the current policy π_θ and the previous policy $\pi_{\theta_{\text{old}}}$), referred to as the *surrogate objective* [55].

[54]: Schulman et al. (2015), *Trust Region Policy Optimization*

[55]: Schulman et al. (2017), *Proximal Policy Optimization Algorithms*

$$
\begin{aligned}
&\underset{\theta}{\text{maximise}}\ \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\,\hat{A}_t\right] \\
&\text{subject to}\ \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}(\cdot \mid s_t)}, \pi_\theta(\cdot \mid s_t)]] \leq \delta
\end{aligned}
$$

Solving this constrained optimisation problem is computationally expensive. The constraint can alternatively be expressed as a penalty β,

resulting in an unconstrained optimisation problem [55].

[55]: Schulman et al. (2017), *Proximal Policy Optimization Algorithms*

$$\underset{\theta}{\text{maximise}}\ \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t - \beta \text{KL}[\pi_{\theta_{\text{old}}(\cdot \mid s_t)}, \pi_\theta(\cdot \mid s_t)] \right]$$

Choosing a single β that performs well for a single or different problems is infeasible [55]. While TRPO outperformed other state-of-the-art algorithms at the time, computing the KL-divergence is expensive, and the implementation is nontrivial.

PPO [55] attempts to solve the shortcomings of TRPO with a *clipped surrogate objective*. The probability ratio of π_θ and $\pi_{\theta_{\text{old}}}$ is denoted as $r_t(\theta)$. TRPO then maximises the loss

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right]$$

PPO proposes a clipping of the probability ratio $r_t(\theta)$ in the loss and thus constraining the policy update:

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

By clipping the loss of $r_t(\theta)$ to the interval $[1 - \epsilon, 1 + \epsilon]$, the incentive to move $r_t(\theta)$ outside of this interval is removed, and thus the policy updates are constrained as well. The hyperparameter ϵ is commonly set to 0.2.

2.7 Multi-agent reinforcement learning

All previous approaches were designed with a single agent operation in mind, but many problems require or benefit from multiple acting agents, as explained in Section 2.1.

A major challenge in MARL is the *non-stationarity* of the environment: From the perspective of a single agent, the environment is constantly changing since the other agents are interacting with the environment as well, resulting in learning stability issues [56] because the Markov assumptions are violated. The *centralised-learning/decentralised execution* paradigm is one possible solution, providing models during training with additional information of the global state that may not be accessible by individual agents due to partial observability [56].

[56]: Lowe et al. (2017), *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments*

Another challenge is the *credit assignment problem* [1]. If multiple agents act simultaneously in the same environment but the environment returns only a single reward signal, the correlation between actions and rewards

[1]: Stone et al. (2000), *Multiagent Systems: A Survey from a Machine Learning Perspective*

decreases, and the learning algorithm has trouble assigning the reward to an action performed by an individual agent.

2.8 Sim-to-real transfer

Real-world data collection is expensive and cumbersome (this is true for both supervised learning and RL), why simulations (such as MuJoCo [17], PyBullet [18] or Habitat [19]) are the preferred way for learning vision models and coordination policies. Even though these simulations have become more realistic and easily accessible in recent years, it is infeasible to model every aspect of real-world physics in simulations. There are three major categories of sources of errors in modelling the real world, which we explain in more detail in the following paragraphs [57].

[17]: Todorov et al. (2012), *MuJoCo: A physics engine for model-based control*

[18]: Coumans et al. (2016), *PyBullet, a Python module for physics simulation for games, robotics and machine learning*

[19]: Savva et al. (2019), *Habitat: A Platform for Embodied AI Research*

[57]: Yu (2024), *On Sim2Real Transfer in Robotics*

Physics Modeling real-world physics is challenging. Even though Newtonian physics simulators have become better over recent years [17, 18], effects such as friction, wheel slip, or actuator backlash remain challenging to model due to complex and intricate interactions on a mechanical level. If a model is trained in simulation, the assumption is that the simulation reflects the real world sufficiently well, but it can be difficult to accurately model the aforementioned effects together with an accurate model of the robot, i.e. a model that has the exact same distribution of masses, centre of gravity, and sensor and actuator positions.

Sensors Next, modelling sensors is challenging. Even though vision simulators have improved over recent years, realistic simulations that capture the appearance of objects in different lights and shadows are still difficult to come by, especially if simulations are expected to be fast (if it takes minutes or even hours to render a frame, then the data might as well come from a real robot). Therefore, simulations often make simplified assumptions, to the extent that simulated images are out-of-distribution from their real-world counterpart. This poses a significant challenge for sim-to-real transfer. This gap is typically smaller in depth sensors [58] since the representation depends less on the appearance of individual objects.

[58]: Loquercio et al. (2021), *Learning high-speed flight in the wild*

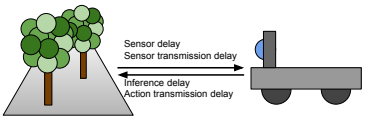


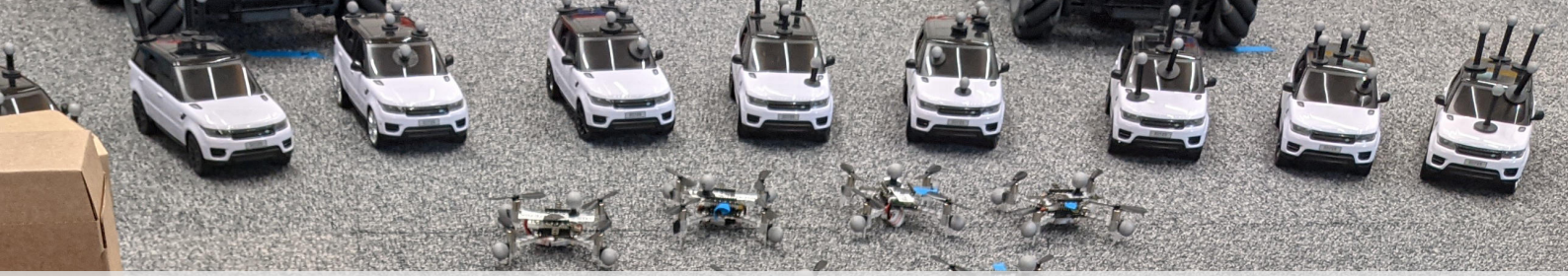
Figure 2.10: Delays occur in many components of a robot system. Sensors process information, information is transmitted, and inference on some models may be run, all incurring a delay.

Delays There are various sources of delays in a robotic system (visualised in Figure 2.10). Sensors have to capture a sample of the environment and may perform some internal processing or compression. The information needs to be transferred to a computer that again performs some processing and requires moving memory. Based on this information, a policy may be invoked, requiring some time for the inference (this could be a learned model or a hand-written controller based on first principles).

Lastly, the computer transfers this information to the actuators, again incurring some transmission delay, and the actuators may take some time to apply the desired force or realise the desired velocity due to the mass and inertia of the robot and the actuators. On top of this, these delays may be inconsistent or vary in duration due to dynamic computing requirements or plain randomness. Any discrepancy in this chain can cause a model trained in simulation to fail when deployed to a robot.

All this results in policies trained in simulation not being able to generalise to the real world, referred to as *reality gap* [5]. The field of *sim-to-real transfer* deals with closing this gap.

[5]: Jakobi et al. (1995), *Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics*



3 Literature review

In the previous chapter, we focused on building the foundations for understanding MAS, MRS, partial observability, decision processes and RL. In this chapter, we conduct a literature review on the state of the art in MARL, methods and approaches in MRS as well as real-world deployments, an in-depth comparison of existing (multi-) robot research platforms, a review on foundation models, and we summarize the state-of-the-art on methods for sim-to-real transfer.

3.1 Multi-agent reinforcement learning 31
3.2 Multi-robot systems . . . 32
3.3 Multi-robot research platforms 34
3.4 Foundation models 35
3.5 Sim-to-real transfer 37
3.5.1 Domain adaptation 37
3.5.2 Domain randomization . 37
3.5.3 Other methods 38

3.1 Multi-agent reinforcement learning

We briefly review work in cooperative multi-agent reinforcement learning, focusing on approaches that model inter-agent communication.

Cooperation enables agents to achieve goals that no individual agent can achieve on its own. However, independently learning agents perform poorly in practice [59] since agents' policies change during training, resulting in a non-stationary environment. Hence, the majority of recent work leans on joint learning paradigms. Q-Mix [60] proposes a central NN combining each agent's value function estimate with other agents' estimates and additional state information. Other agents' actions are taken into consideration during training. Counterfactual multi-agent policy gradients (COMA) [61], MA-Deep deterministic policy gradient (DDPG) [56] or TS-TRPO [62] use a policy-gradient approach with a *centralised critic* (a centralised value function predictor) and address the credit assignment problem with an efficient counterfactual baseline. These approaches avoid the need for explicit communication by making strong assumptions about the visibility of other agents and the environment.

[59]: Matignon et al. (2012), *Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems*

[60]: Rashid et al. (2018), *QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning*

[61]: Foerster et al. (2018), *Counterfactual Multi-Agent Policy Gradients*

[56]: Lowe et al. (2017), *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments*

[62]: Gupta et al. (2017), *Cooperative Multi-agent Control Using Deep Reinforcement Learning*

Most learning algorithms follow the *centralised training, decentralised execution* paradigm to a certain extent. This paradigm was first introduced for centralised planning [63] and typically means that the agents and value functions have global information and state knowledge and full connectivity during training, which can be ensured when training in simulation, and thus eases training.

[63]: Kraemer et al. (2016), *Multi-agent reinforcement learning as a rehearsal for decentralized planning*

Early work on learning communication considers *discrete* communication through signal binarization [64], or categorical communication emissions [65]. The former approaches demonstrate emergent communication

[64]: Foerster et al. (2016), *Learning to Communicate with Deep Multi-Agent Reinforcement Learning*

[65]: Mordatch et al. (2018), *Emergence of grounded compositional language in multi-agent populations*

among a few agents; scaling the learning process to larger agent teams requires innovations in the structure of the learnt communication models. The approach enables speaker agents to output semantic information, which listener agents then observe. This type of communication is also not differentiable, and therefore learnable, and assumes time-invariant fully connected agent topologies. Some other approaches use a predetermined communication protocol, for instance [66, 67].

[66]: Maravall et al. (2011), *Coordination of Communication in Robot Teams by Reinforcement Learning*
[67]: Zhang et al. (2013), *Coordinating Multi-Agent Reinforcement Learning with Limited Communication*

3.2 Multi-robot systems

In this section, we review related MRS testbeds and frameworks. Our survey includes centralised frameworks and decentralised methods that use machine-learning-based approaches or communication. We proceed to review the related work on robotic communication frameworks and standards to evaluate an appropriate choice for our use case.

Multi-robot applications MRS necessitate the coordination of multiple, often fast-moving, robots within proximity in real-world settings. In flocking, agile agents maintain proximity using rules of separation, alignment, and cohesion [68]. It is believed that flocks of birds use a similar set of simple rules that collectively show impressive emergent behaviour. In formation control, agents maintain or adjust formations while navigating around obstacles [69]. In trajectory deconfliction tasks, multiple agents have a designated goal and must avoid collisions with other agents dynamically (Reciprocal velocity obstacles (RVO)) [16]. Multi-agent path planning has a similar objective to RVO, but paths are planned ahead of time instead of reactive behaviour. One example is Conflict-based search (CBS) [70]. In multi-agent object manipulation [71, 72], teams of agents have to coordinate to manipulate a dynamic object in the environment. In area coverage [73, 74], multiple agents have to scatter in the environment to cover an area, e.g. for remote sensing.

[68]: Reynolds (1987), *Flocks, Herds and Schools: A Distributed Behavioral Model*

[69]: Oh et al. (2015), *A survey of multi-agent formation control*

[16]: Van den Berg et al. (2008), *Reciprocal velocity obstacles for real-time multi-agent navigation*

[70]: Sharon et al. (2015), *Conflict-based search for optimal multi-agent pathfinding*
[71]: Martinoli et al. (2004), *Modeling swarm robotic systems: A case study in collaborative distributed manipulation*
[72]: Alonso-Mora et al. (2015), *Local motion planning for collaborative multi-robot manipulation of deformable objects*
[73]: Zelinsky et al. (1993), *Planning paths of complete coverage of an unstructured environment by a mobile robot*
[74]: Smith et al. (2012), *Persistent Robotic Tasks: Monitoring and Sweeping in Changing Environments*

Real-world multi-robot deployments Decentralised miniature autonomous drones have been deployed to a variety of real-world test cases that use a trajectory planner enabling efficient, obstacle-avoiding, and swarm-coordinated navigation in highly cluttered environments using onboard sensors [75]. A decentralised flocking model with 30 real drones is presented in [76], incorporating an evolutionary optimisation framework to handle constrained motion, communication, and obstacles. The Swarm gradient bug algorithm (SGBA) enables tiny, autonomous flying robots to explore unknown indoor environments, avoid obstacles, communicate to avoid collisions, and return to their departure point without external infrastructure [77]. A vision-based detection and tracking algorithm that enables drone swarms to navigate without inter-agent communication

[75]: Zhou et al. (2022), *Swarm of micro flying robots in the wild*
[76]: Vásárhelyi et al. (2018), *Optimized flocking of autonomous drones in confined environments*
[77]: McGuire et al. (2019), *Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment*

or visual markers, using a CNN and multi-agent state tracking for real-time flocking control, is presented in [78] and validated by outdoor experiments. Collective behaviours in a swarm of fish-inspired miniature underwater robots are presented in [79]. The robots rely on implicit communication through blue light sensing, achieving complex behaviours like synchrony, dispersion, and dynamic formations without centralised control. Micro-robots with self-organising behaviours in a swarm of 300 robots that rely on local interactions to autonomously form organic and adaptable shapes are presented in [80]. Similarly, programmable self-assembly of complex two-dimensional shapes using a thousand-robot swarm, enabled by autonomous robots cooperating through local interactions, is demonstrated in [81].

[78]: Schilling et al. (2021), *Vision-Based Drone Flocking in Outdoor Environments*

[79]: Berlinger et al. (2021), *Implicit coordination for 3D underwater collective behaviors in a fish-inspired robot swarm*

[80]: Slavkov et al. (2018), *Morphogenesis in robot swarms*

[81]: Rubenstein et al. (2014), *Programmable self-assembly in a thousand-robot swarm*

Robotic communications frameworks Communication 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 [82], with various communications frameworks overlaid on top of these low-level technologies (e.g. Real time publish subscribe (RTPS), Message queuing telemetry transport (MQTT) [83] or standard Internet protocol (IP) [84]). Whatever the specific technology, the underlying protocol suites and the nature of wireless communication set fundamental limitations [85] on available messaging rates when multiple agents communicate in a decentralised manner. Multiple strategies exist that attempt to maximise protocol performance under specific conditions [86, 87], including dynamic centralisation using homogeneous agents [88]. Despite these strategies, these systems' performance at scale still needs to be better tested in real-world robotics systems, which often entail unexpected overheads [89].

[82]: Frotzscher et al. (2014), *Requirements and current solutions of wireless communication in industrial automation*

[83]: Tran et al. (2018), *Intelligent robotic IOT system (IRIS) testbed*

[84]: Stubbs et al. (2006), *Multivehicle systems control over networks: a hovercraft testbed for networked and decentralized control*

[85]: Wang et al. (2006), *Performance Limits and Analysis of Contention-based IEEE 802.11 MAC*

[86]: Bhargava et al. (2018), *Improve Collision in Highly Dense WiFi Environment*

[87]: Gielis et al. (2021), *Improving 802.11p for Delivery of Safety-Critical Navigation Information in Robot-to-Robot Communication Networks*

[88]: Kang et al. (2019), *RECEN:Resilient MANET Based Centralized Multi Robot System Using Mobile Agent System*

[89]: Kronauer et al. (2021), *Latency Analysis of ROS2 Multi-Node Systems*

Graph neural networks for multi-robot control Recent advances use GNNs for enhanced control and perception, enabling inter-robot communication through latent messages to address the challenge of partial observability. Early results focus on the study of flocking [4, 90]. These works develop a distributed controller for large networks of simulated mobile robots that use local information and communication by imitating centralised first-principles flocking controllers during training.

Later, the application to MAPF has raised the interest of researchers [3, 91]. The authors propose a model that synthesises local communication and decision-making policies for decentralised multi-robot path planning in constrained workspaces and demonstrate its performance in simulations with results close to an expert algorithm, and the ability to generalise to larger environments and teams. Lastly, GNNs have been applied to cooperative perception to aggregate and share local information observed

[4]: Gama et al. (2021), *Graph Neural Networks for Decentralized Controllers*

[90]: Tolstaya et al. (2020), *Learning Decentralized Controllers for Robot Swarms with Graph Neural Networks*

[3]: Li et al. (2020), *Graph neural networks for decentralized multi-robot path planning*

[91]: Gama et al. (2022), *Synthesizing Decentralized Controllers With Graph Neural Networks and Imitation Learning*

from cameras or other sensors with autonomous cars [92, 93]. VGAI is the first to combine cooperative perception with first-person visual information with control by imitating a flocking policy in a swarm of drones using GNNs [94].

[92]: Wang et al. (2020), *V2VNet: Vehicle-to-Vehicle Communication for Joint Perception and Prediction*
[93]: Zhou et al. (2022), *Multi-Robot Collaborative Perception With Graph Neural Networks*
[94]: Hu et al. (2021), *VGAI: End-to-end learning of vision-based decentralized controllers for robot swarms*

3.3 Multi-robot research platforms

In the last decade, the research community has been introduced to a wide range of (multi-) robot platforms. This in-depth literature review of commercial and research platforms presents an overview of existing platforms. The classification scheme differentiates between them based on their dynamics (Ackermann, differential, omnidirectional drive, airborne/multirotor platforms) and their specific use cases in single- or multi-agent research. Since we are primarily interested in multi-robot research, we investigate the tradeoff between the cost and speed of these platforms, as agility is an attribute we require in the team (note that we use 'speed' and 'agility' interchangeably – this is a justified simplification for indoor robots operating in confined spaces). Cost limits the size of the team.

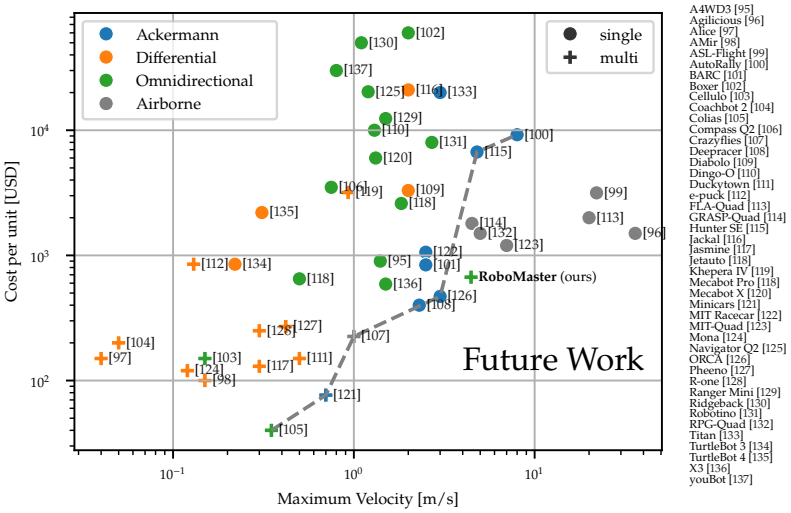


Figure 3.1: We compare various research and commercial robotic platforms concerning their tradeoff between cost per unit and maximum velocity. We differentiate between Ackermann, Differential, Airborne, and Omnidirectional platforms and between platforms specifically targeting multi-agent research. A clear correlation between speed and cost can be seen, as well as a trend for multi-agent platforms to be low-cost but, as a consequence, slow. We draw the Pareto front for non-airborne platforms as a dashed line, with work introduced in this thesis pushing the limits on the cost and speed tradeoff. We show the airborne platforms as a reference as we are interested in ground-based platforms only.

Table 3.1: We list critical specifications for omnidirectional robots from related work. 'Multi-Agent' indicates whether the platform is introduced as a multi-agent platform, and 'Commercial' indicates whether the platform can be obtained commercially.

Brand/Uni	Product	Vel [m/s]	Cost [USD]	Mass [kg]	L x B x H [mm]	Multi Agent	Commercial
EPFL	Cellulo [103]	0.15	150.0	0.18	73 × 80 × 80	✓	
Lincoln Uni	Colias [105]	0.35	40.0	0.01	40 × 40 × 40	✓	
Hiwonder	Jetauto [118]	0.50	650.0	4.00	324 × 659 × 260		✓
Hangfa	Compass Q2 [106]	0.75	3,500.0	16.00	430 × 330 × 115		✓
KUKA	youBot [137]	0.80	30,000.0	20.00	580 × 380 × 140		✓
Clearpath	Ridgeback [130]	1.10	50,000.0	135.00	790 × 960 × 310		✓
Hangfa	Navigator Q2 [125]	1.20	20,300.0	35.00	520 × 480 × 240		✓
Clearpath	Dingo-O [110]	1.30	10,000.0	13.00	680 × 510 × 111		✓
RoboWorks	Mecabot X [120]	1.32	6,000.0	20.50	630 × 581 × 203		✓
Lynxmotion	A4WD3 [95]	1.40	900.0	5.55	372 × 374 × 152		✓
RDK	X3 [136]	1.50	590.0	1.93	181 × 236 × 185		✓
Agilex	Ranger Mini [129]	1.50	12,400.0	64.50	500 × 738 × 338		✓
RoboWorks	Mecabot Pro [118]	1.83	2,600.0	10.80	541 × 581 × 225		✓
Clearpath	Boxer [102]	2.00	60,000.0	127.00	550 × 750 × 340		✓
Festo	Robotino [131]	2.70	8,000.0	22.80	450 × 450 × 325		✓
Cambridge	**RoboMaster** (ours)	4.45	670.0	3.00	240 × 320 × 200	✓	✓

Figure 3.1 visualises our survey in a 2D plot, and Table 3.1 provides detailed statistics specifically for for omnidirectional drive robots. We find that the majority of multi-agent research platforms utilise differential drive [97, 98, 111, 112, 117, 119, 124, 127, 128], thus positioning them in the bottom left corner of Figure 3.1. The reason is that such platforms tend to be designed to minimise cost, with 10s of such agents in mind. The slowest of these platforms is Alice [97] at 4 cm/s, and the fastest is the Khepera IV [119] with 0.93 m/s. Three ground-based platforms are specifically geared towards multi-agent research but use dynamics other than differential drive, specifically the Minicar [121], Cellulo[103], and Colias [105]. More recently, slightly faster commercial differential drive robots have become popular [109, 116, 134, 135], ranging from 0.22 m/s for the Turtlebot 3 [134] to 2 m/s for the Diabolo [109].

[97]: Caprari et al. (2003), *Design and control of the mobile micro robot alice*

[119]: Soares et al. (2016), *The Khepera IV mobile robot: performance evaluation, sensory data and software toolbox*

[121]: Hyldmar et al. (2019), *A Fleet of Miniature Cars for Experiments in Cooperative Driving*

[103]: Özgür et al. (2017), *Cellulo: Versatile Handheld Robots for Education*

[105]: Arvin et al. (2014), *Development of an autonomous micro robot for swarm robotics*

From the bottom left, we can see two clusters emerging towards the top and the right. The omnidirectional drive robots [95, 102, 103, 105, 106, 110, 118, 120, 125, 129–131, 136, 137] tend to move faster than differential drive, ranging from 0.15 m/s (Cellulo [103]) to 2.7 m/s (Festo Robotino [131]), but are also significantly more expensive, with a mean cost of USD 13k.

[131]: FESTO Robotino (2024), *FESTO Robotino*

Aerial platforms (such as multi-rotors), on the other hand, are moderately priced at a mean of USD 1600 and can move at much higher speeds of up to 36 m/s (Agilicious [96]), but come with other disadvantages compared to ground robots, such as operational overheads like short runtime (minutes as opposed to hours) or more challenging infrastructure needs (recovery and safety mechanisms and the availability of accurate indoor tracking). These issues become particularly noticeable and more pronounced when developing multi-agent systems – the feasibility of an experiment depends on all robots operating without faults and risks concurrently. In a MRS, individual robots requiring additional operator attention and overhead can thus pose a significant barrier to rapid deployment and testing.

[96]: Foehn et al. (2022), *Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight*

To the right of the plot, we observe a cluster of Ackermann drive robots [100, 101, 108, 115, 121, 122, 126, 133]. These platforms move from top speeds ranging from 0.7 m/s for the Minicar [121] (USD 76.5) to 8 m/s for the AutoRally [100] (ca. USD 10k), with an average of USD 4850. While some of these platforms are agile, they tend to be expensive and unsuitable for constrained indoor spaces.

[100]: Williams et al. (2016), *Aggressive driving with model predictive path integral control*

3.4 Foundation models

Foundation models are large neural networks typically trained on broad data, using self-supervised learning at scale, that can be adapted to a wide range of downstream tasks. Their significance can be summarised through *emergence*, meaning that the behaviour of a system is induced

implicitly instead of trained explicitly, and *homogenisation* refers to the capability of methods that allow building learning systems for a wide range of applications [138].

[138]: Bommasani et al. (2021), *On the Opportunities and Risks of Foundation Models*

In the following, we perform a literature review for the four classes of foundation models: LLMs, Vision Models, Vision and language model (VLM), and Vision, language and action models (VLAs). State-of-the-art foundation models typically make use of the transformer architecture [139].

[139]: Vaswani et al. (2017), *Attention is All you Need*

The trend of training large models self-supervised on a large body of data started in Natural language processing (NLP). Each word or token is associated with a context-independent vector. BERT [140] and GPT-2 [141] were the first models trained this way and marked the inflexion point in NLP at which the research community as a whole pivoted towards using a single model for a wide range of different tasks. Later models significantly scaled up the number of parameters, thus showing the continuance of emergence and homogeneity with an increased model capacity (e.g. GPT-3 [142] with 175 billion parameters and GPT-4 [143] with 1.800 trillion, compared to 1.5 billion in GPT-2).

[140]: Devlin et al. (2019), *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*
[141]: Radford et al. (2019), *Language models are unsupervised multitask learners*

[142]: Brown et al. (2020), *Language Models are Few-Shot Learners*
[143]: Achiam et al. (2023), *Gpt-4 technical report*

Soon after the boom on LLMs, researchers applied self-supervised learning on large datasets with transformers on other modalities. For instance, Dino [144] and DinoV2 [145] focus on the application of images. Building on this, multi-modal models became popular, combining images and textual prompts. Exambles include CLIP [146] DALL-E [147] and DALL-E-v2 [148], and Stable Diffusion [149]. Lastly, robotics foundation models, or VLAs, expand this paradigm to include robot actions. Examples include RT-1 [150], RT-2 [151] or OpenVLA [152].

[145]: Oquab et al. (2023), *DINOv2: Learning Robust Visual Features without Supervision*
[146]: Radford et al. (2021), *Learning Transferable Visual Models From Natural Language Supervision*
[148]: Ramesh et al. (2022), *Hierarchical text-conditional image generation with clip latents*
[149]: Rombach et al. (2022), *High-resolution image synthesis with latent diffusion models*
[152]: Kim et al. (2024), *OpenVLA: An Open-Source Vision-Language-Action Model*

Recently, a trend for open-source foundation model has been popularised by private research institutions, leading to the emergence of the Llama series of LLMs (Llama [153], Llamav2 [154] and Llamav3 [155]). Similarly, DinoV2 [145] and OpenVLA [152] are open source and even free for commercial applications.

[153]: Touvron et al. (2023), *Llama: Open and efficient foundation language models*
[154]: Touvron et al. (2023), *Llama 2: Open Foundation and Fine-Tuned Chat Models*
[155]: Dubey et al. (2024), *The Llama 3 Herd of Models*

All previously mentioned foundation models have a large number of model parameters in common (typically in the order of tens of millions to a billion). Assuming these models are represented as 16-bit float, loading them requires hundreds of megabytes to gigabytes of working memory. This number does not include additional memory for the data, which depending on the mechanism used, can increase quadratic (in the case of the attention mechanism [139]). This, as well as the latency required to run a forward pass in this model (as further explained in the sim-to-real challenges in Section 2.8), makes an immediate unprocessed deployment on robots requiring real-time processing often infeasible or challenging, thus requiring methods such as model distillation [156] for smaller models and quantisation for a smaller memory footprint [157].

[156]: Hinton et al. (2015), *Distilling the Knowledge in a Neural Network*
[157]: Jacob et al. (2018), *Quantization and training of neural networks for efficient integer-arithmetic-only inference*

3.5 Sim-to-real transfer

After explaining the challenges of sim-to-real transfer in Section 2.8, we now review methods promising to narrow or close the reality gap.

3.5.1 Domain adaptation

(Visual) domain adaptation [158] (also exemplified in [159–162]) is a technique to align a policy trained in the source domain to the target domain, even though these two domains differ. It is, therefore, commonly used for the sim-to-real transfer, where the source domain is the simulation and the real-world domain is the target domain.

[158]: Long et al. (2015), *Learning Transferable Features with Deep Adaptation Networks*

In visual domain adaptation, we differentiate between *feature-level adaptation*, which learns domain-invariant features, and *pixel-level adaptation*, which makes images from one domain look similar to images from another domain. A disadvantage of domain adaptation is that it generally requires large datasets. Even though these datasets do not have to be labelled explicitly (unsupervised), it can be hard to come by the required unbiased data in robotics tasks – examples for visual domain adaptation for robotics and sim-to-real transfer are [163, 164].

[163]: Tzeng et al. (2020), *Adapting Deep Visuomotor Representations with Weak Pairwise Constraints*

[164]: Stein et al. (2018), *Genesis-rt: Generating synthetic images for training secondary real-world tasks*

3.5.2 Domain randomization

Domain Randomization [6] (also exemplified in [165–168]) attempts to randomise the simulation such that the real world can be considered *one possible random permutation of the diversified simulation* by adding random noise and permutations. While this seems intuitive, it makes the task more challenging than necessary, leading to suboptimal policies. *Active domain randomisation* attempts to counteract this issue by making the sampling adaptive instead of uniform, thus leading to a high variance in the policy actions [169]. While this approach has successfully been applied to many problems in supervised learning, there is some evidence that domain randomisation destabilises RL [167, 170].

[6]: Tobin et al. (2017), *Domain randomization for transferring deep neural networks from simulation to the real world*

[169]: Mehta et al. (2020), *Active domain randomization*

[167]: Matas et al. (2018), *Sim-to-real reinforcement learning for deformable object manipulation*

[170]: Zhang et al. (2019), *VR-Goggles for Robots: Real-to-Sim Domain Adaptation for Visual Control*

Data augmentation is closely related to domain randomisation and avoids overfitting by applying cropping, flipping, and photometric variations to images [171, 172].

[171]: Krizhevsky et al. (2012), *ImageNet Classification with Deep Convolutional Neural Networks*

[172]: Fan et al. (2021), *SECANT: Self-Expert Cloning for Zero-Shot Generalization of Visual Policies*

Domain randomisation is commonly used in robotics applications by randomising aspects of a training simulation environment for grasping and other manipulation tasks [6, 165–168, 173] and to vary physical properties such as friction [174], mass distribution and controller parameters [175–178], or delays [179].

3.5.3 Other methods

In this section, we briefly review other sim-to-real transfer methods.

Sim-to-real via modularity and abstraction: Even though simulators have improved over recent years, they will not be indistinguishable from the real world. Most other sim-to-real methods attempt to close the reality gap explicitly in end-to-end systems. It has been demonstrated that a good choice of hand-crafted abstractions and interfaces in a learning system can improve sim-to-real transfer significantly compared to systems trained end-to-end [180]. This is also referred to as *zero-shot sim-to-real transfer*.

[180]: Müller et al. (2018), *Driving policy transfer via modularity and abstraction*

Sim-to-real via sim-to-sim: A significant disadvantage of domain adaptation is the necessity for a large amount of unlabeled real-world training data, which can be hard to come by depending on the problem, and domain randomisation wastes modelling power and expressiveness. It has been proposed to train a randomised-to-canonical adaption function that is learned for an interpretable intermediate representation by first performing randomisations in simulation and then learning a model that maps the randomised images back to the source domain [181].

[181]: James et al. (2019), *Sim-To-Real via Sim-To-Sim: Data-Efficient Robotic Grasping via Randomized-To-Canonical Adaptation Networks*

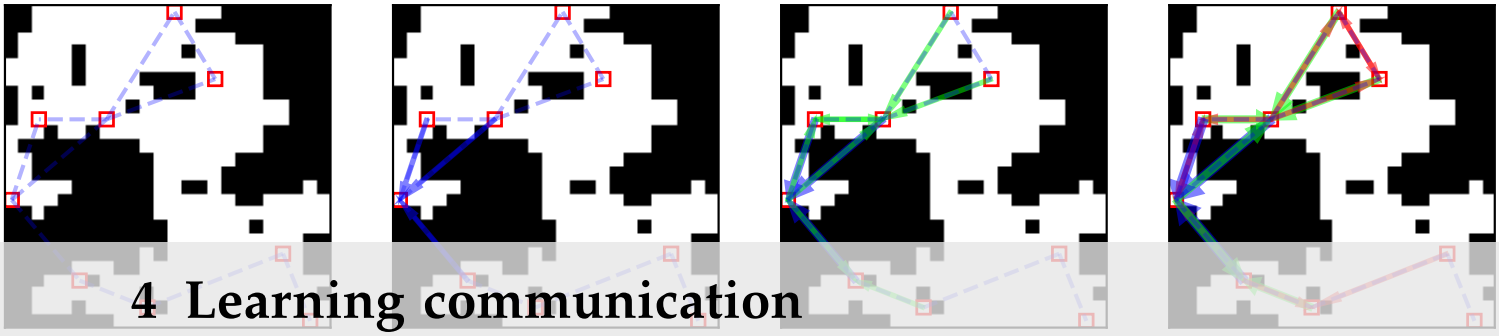
Sim-to-real via real-to-sim: Similarly to *sim-to-real via sim-to-sim*, we can learn an adaptation function that translates real-world samples to equivalent simulation samples and use the policy trained in simulation together with this adaptation model, thus decoupling the two domains. This allows a specific adaptation model for different real-world environments and adapts real-world images to resemble simulated ones, thus decoupling the two domains. Therefore, if the real-world environment changes, only the translator has to be re-trained. Image-to-image translation has been used for tactile sensors [182] or robot control [170] for sim-to-real transfer by transferring images from the real world to the simulation domain, often using architectures from the CycleGAN [183] family.

[182]: Isola et al. (2017), *Image-to-Image Translation with Conditional Adversarial Networks*
[170]: Zhang et al. (2019), *VR-Goggles for Robots: Real-to-Sim Domain Adaptation for Visual Control*
[183]: Zhu et al. (2017), *Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks*

Sim-to-real via real-world training: Lastly, sim-to-real transfer can be achieved by training or tuning models in the real world directly on the robots. A significant challenge is to avoid damage to the robots, which is why augmented or mixed-reality settings are particularly attractive: A single physical robot is interacting with the environment, while other virtual robots running the same policy are simulated [184]. This way, collisions can be penalised without inflicting damage to the robot.

[184]: Mitchell et al. (2020), *Multi-Vehicle Mixed Reality Reinforcement Learning for Autonomous Multi-Lane Driving*

Learning communication for multi-robot coordination



4 Learning communication

This thesis's overarching ultimate goal is deploying capable MRS to the real world. An essential building block is training multi-agent coordination policies for such MRS. One way to achieve this goal is learning from environmental interactions, which is particularly useful if the optimal strategy is unknown.

MARL models arise as natural solutions to problems where the joint actions of multiple decision-making agents influence a common environment. In decentralised systems, agents not only need to learn how to behave cooperatively but also how to *communicate* to most effectively coordinate their actions in pursuit of a common goal. Even though effective communication is key to successful decentralised coordination, how a problem benefits from communication is not necessarily predetermined, especially in complex multi-agent settings where the optimal strategy is unknown. RL has become one of the most promising avenues for solving such problems.

Capturing information that enables complex inter-agent coordination requires new kinds of NN architectures. GNNs exploit that inter-agent relationships can be represented as graphs. In MAS, an agent is modelled as a node in the graph, the connectivity of agents as edges, and the internal state of an agent as a graph signal. Recently, a range of approaches towards learning explicit communication were made [3, 64, 90, 185]. The key attribute of GNNs is that they operate locally, whereby information is shared over a multi-hop communication network through explicit communication with nearby neighbours only, resulting in decentralised policies.

In this chapter, we fill a gap in MARL algorithms in which agents learn to share local observations in a team of cooperative agents, each of which obtains local rewards (thus alleviating the credit assignment problem), contributing to some overarching team objectives. This work is foundational for training cooperative multi-robot coordination policies based on GNNs and allows end-to-end training of decentralisable multi-robot coordination policies for arbitrary multi-robot control problems. To test our algorithm, we study the coverage path planning problem, in which multiple agents must act cooperatively to visit every tile in a discrete grid-world as quickly as possible. We provide the theoretical foundation based on the policy-gradient class of RL algorithms and

4.1 Contributions 42
4.2 Approach 42
4.2.1 Independent vanilla policy gradient 42
4.2.2 Cooperative learning . . . 44
4.3 Experiments 46
4.3.1 Environment setup 47
4.3.2 Non-convex coverage . . . 47
4.3.3 Split coverage 48
4.3.4 Path planning 48
4.4 Results 49
4.4.1 Metrics 49
4.4.2 Baseline 50
4.4.3 Performance evaluation . 50
4.4.4 Generalization 52
4.4.5 Policy interpretation . . . 54
4.5 Conclusion 58

[3]: Li et al. (2020), *Graph neural networks for decentralized multi-robot path planning*
[64]: Foerster et al. (2016), *Learning to Communicate with Deep Multi-Agent Reinforcement Learning*
[90]: Tolstaya et al. (2020), *Learning Decentralized Controllers for Robot Swarms with Graph Neural Networks*
[185]: Prorok (2018), *Graph Neural Networks for Learning Robot Team Coordination. Federated AI for Robotics Workshop*

provide proof of optimality. We proceed with the empirical results of our approach on three scenarios and evaluate the scalability of our trained policies to up to 30 agents. We also analyse the communication contents and patterns based on two different methods. This chapter is based on [21] and [7].

[21]: Blumenkamp (2020), *Learning interpretable explicit communication for the multi-agent coverage problem*

[7]: Blumenkamp et al. (2021), *The Emergence of Adversarial Communication in Multi-Agent Reinforcement Learning*

Supplementary material

We provide the following supplementary material:

Code: Talk: Video:

4.1 Contributions

Our core contributions of this chapter are:

- We develop a new foundational multi-agent learning model that integrates policies that share a learnable communication channel, enabling the training of end-to-end MARL policies for multi-robot control. This model consists of two key components, *(i)* a monolithic, decentralisable neural architecture that accommodates multiple distinct reward functions and a common differentiable communication channel provided through GNNs, *(ii)* an RL algorithm that elicits the emergence of cooperative communications from local rewards.
- We perform an experimental evaluation based on a MAS of cooperative agents and demonstrate the effectiveness of the learning scheme in multi-agent coverage and path planning problems. Our results show that it is possible to learn effective communication strategies. Our emphasis is on general cooperative multi-agent learning paradigms that build the foundation for later chapters of this thesis, rather than improving the performance of state-of-the-art coverage methods.

4.2 Approach

This chapter considers the presence of an explicit communication channel between agents. For an introduction to GNNs, please refer to Section 2.3.2. For an introduction to decision processes, please refer to Section 2.4. For an introduction to RL, please refer to Section 2.5, and for an introduction to deep RL, refer to Section 2.6. We first introduce a variation of Vanilla policy gradient (VPG) [186] for groups of independent agents. We then build on top of this to extend this to a multi-agent learning algorithm capable of learning communication and provide proof for optimality.

[186]: Schulman et al. (2016), *High-Dimensional Continuous Control Using Generalized Advantage Estimation*

4.2.1 Independent vanilla policy gradient

We consider MAS that are partially observable and operate decentralised. Each agent aims to maximise a local reward without access to the true

global state. This builds the foundation for Section 4.2.2, where we allow agents to communicate through the exchange of explicit messages. We model this as a POMG.

Multi-agent vanilla policy gradient (VPG) We detail VPG applied to independent agents with a centralised critic. This is a setting similar to COMA [61] but with individual agent-specific rewards. Please refer to Section 2.6 for more details. Without explicit communication, each agent learns a local policy

[61]: Foerster et al. (2018), *Counterfactual Multi-Agent Policy Gradients*

$$\pi^i_{\theta^i}(a^i_t|z^i_t) \ : \mathcal{Z} \times \mathcal{A} \rightarrow [0,1]$$

parameterised by a local set of parameters θ^i that maps an observation z^i_t of the observation space $\mathcal{Z}$ and an action a^i_t from the action space $\mathcal{A}$ to a likelihood (an alternative formulation maps observation directly to action). The induced joint policy is

$$\boldsymbol{\pi}_\theta(\mathbf{a}_t|\mathbf{z}_t) = \prod_{R^i \in \mathcal{V}} \pi^i_{\theta^i}(a^i_t|z^i_t)$$

with $\theta = \{\theta^i\}_{R^i \in \mathcal{V}}$. The discounted return for each agent R^i is

$$G^i_t = \sum_{l=0}^{\infty} \gamma^l r^i_{t+l}.$$

The centralised value functions that estimate the value of each agent R^i in state s_t are

$$V^{\boldsymbol{\pi},i}(s_t) = \mathbb{E}_t[G^i_t|s_t],$$

and the corresponding action-value functions are

$$Q^{\boldsymbol{\pi},i}(s_t, \mathbf{a}_t) = \mathbb{E}_t[G^i_t|s_t, \mathbf{a}_t]$$

(we now omit $\boldsymbol{\pi}$ and simply write V^i and Q^i). The advantage functions are then given by

$$A^i = Q^i(s_t, \mathbf{a}_t) - V^i(s_t).$$

Many techniques exist to estimate the advantage. For example, the *Generalised Advantage Estimate* is defined as

$$\hat{A}^i = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta^i_{t+l}$$

where $\lambda \in [0,1]$ and where the TD-residual is

$$\delta^i_t = r^i_t + \gamma V^i(s_{t+1}) - V^i(s_t),$$

as introduced in [186]. The value functions themselves can be estimated

[186]: Schulman et al. (2016), *High-Dimensional Continuous Control Using Generalized Advantage Estimation*

by parameterised functions V^i_ϕ (with parameters ϕ) by minimizing

$$\mathbb{E}_t[\|V^i_\phi(s_t) - G^i_t\|^2].$$

However, since this is not the focus of this chapter, we will now assume that we have access to a centralised advantage estimate $\hat{A}^i$ for each agent R^i. Finally, the policy of agent R^i can be improved through gradient ascent using the policy gradient

$$g^i = \mathbb{E}_t \left[\nabla_\theta \log \pi^i_{\theta^i}(a^i_t|z^i_t)\hat{A}^i\right].$$

4.2.2 Cooperative learning

We consider a team of cooperative homogeneous agents with individual rewards and explicit communication and formulate a local decentralised policy. We modify VPG to train the team of cooperative agents to collaborate using explicit communication. An overview of the architecture is illustrated in Figure 4.1 and Figure 4.2.

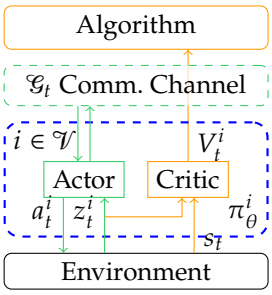


Figure 4.1: Architectural overview: The actor or the policy π^i_θ determines the action a^i_t from the local observation z^i_t and the multi-hop communication messages from other agents, which in turn depend on all agents' observations z_t and the communication topology defined through the graph $\mathcal{G}_t$. Green denotes components that can be executed locally, and orange denotes components required for centralised training only.

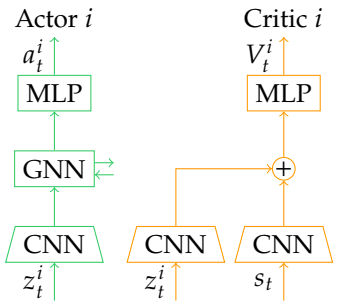


Figure 4.2: Close-up of actor and critic architecture: The actor uses local information z^i_t, encoded with a CNN and communicated through a GNN to and from other agents to predict an action a^i_t. The critic has additional access to privileged global state information s_t and, therefore, does not need to rely on the GNN for communication.

Local Policy To train a policy that utilises explicit communication, we must first define a model architecture that facilitates differentiable communication. As introduced in Section 2.3.2, an MRS can be represented as graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with robots representing the node set and communication links between robots the edge set. The neighbourhood $\mathcal{N}_i$ of node $R^i \in \mathcal{V}$ is defined through a static communication range d. $\mathcal{N}_i$ includes node R^i.

Each agent encodes its local observation z^i_t using an encoder

$$\nu^{\text{obs}}(z^i_t) \; : \; \mathcal{Z} \to \mathbb{R}^F$$

(e.g., using a CNN). Furthermore, there is a global state encoder

$$\nu^{\text{state}}(s_t) \; : \; \mathcal{S} \to \mathbb{R}^G$$

that guides the value prediction with privileged centralised information only available during training.

The k-th message (or *datum*) sent by agent R^i at time t is $\mathbf{h}^i_{k,t}$, where k can be interpreted as a sub-time-step, as the messages are generated from multiple hops of communication at the same policy time step t. Practically, K messages would be sent between t and $t+1$. The embedding $\nu^{\text{obs}}(z^i_t) = \mathbf{h}^i_{0,t}$ is the first message and input to a message-passing GNN with K layers built from the following base layer:

$$\mathbf{h}^i_{k+1,t} = \phi^{\text{post}}_k \left(\frac{1}{|\mathcal{N}_i|} \sum_{R^j \in \mathcal{N}_i} \phi^{\text{pre}}_k(\mathbf{h}^j_{k,t}) \right). \tag{4.1}$$

The action is computed from $\mathbf{h}^i_{K,t}$ as

$$a^i_t = \psi_{\text{act}}(\mathbf{h}^i_{K,t})$$

and the value function

$$V^{\pi,i}(s_t) = \psi^{\text{val}}(\nu^{\text{state}}(s_t), \mathbf{h}^i_{K,t})$$

with

$$\theta = \{\nu^{\text{state}}, \nu^{\text{obs}}, \phi^{\text{pre}}_1, \phi^{\text{post}}_1, \ldots, \phi^{\text{pre}}_K, \phi^{\text{post}}_K, \psi^{\text{act}}, \psi^{\text{val}}\}.$$

Despite being decentralised and locally executable during inference (as there is no need to predict the value function that depends on global state information during evaluation time), each agent's policy explicitly depends on the observations made by neighbouring agents due to the explicit communication.

Cooperative policy gradient To train the cooperative group of agents, we modify VPG (Section 4.2.1) to reinforce local, individual actions that lead to increased rewards for other agents. We assume the availability of an explicit communication channel between agents.

Lemma 4.2.1 *Given an actor-critic algorithm with a compatible TD(1) critic that follows the cooperative policy gradient*

$$g^i_k = \mathbb{E}_\pi \left[\sum_{R^j \in \mathcal{V}} \nabla_\theta \log \pi^j_\theta(a^j|z^j) \hat{A}^i \right] \tag{4.2}$$

for each agent $R^i \in \mathcal{V}$ at each iteration k, this gradient converges to a local maximum of the expected sums of returns of all agents with probability one.

This lemma highlights that the gradient converges and that the joint policy maximises the sum of cumulative rewards.

Proof. The total gradient applied to θ is given by

$$\begin{aligned}
g &= \mathbb{E}_\pi \left[\sum_{R^i \in \mathcal{V}} \sum_{R^i \in \mathcal{V}} \nabla_\theta \log \pi^j_{\theta^j}(a^j|z^j) \hat{A}^i \right] \\
&= \mathbb{E}_\pi \left[\sum_{R^i \in \mathcal{V}} \hat{A}^i \sum_{R^i \in \mathcal{V}} \nabla_\theta \log \pi^j_{\theta^j}(a^j|z^j) \right] \\
&= \mathbb{E}_\pi \left[\sum_{R^i \in \mathcal{V}} \hat{A}^i \nabla_\theta \log \prod_{R^j \in \mathcal{V}} \pi^j_{\theta^j}(a^j|z^j) \right] \\
&= \mathbb{E}_\pi \left[\sum_{R^i \in \mathcal{V}} \hat{A}^i \nabla_\theta \log \boldsymbol{\pi}_\theta(\mathbf{a}_t|\mathbf{z}_t) \right].
\end{aligned}$$

If we consider the sum of rewards $r_t = \sum_{R^i \in \mathcal{V}} r_t^i$ as the joint reward obtained by the joint policy $\boldsymbol{\pi}$, the joint advantage estimate is $\hat{A}_t = \sum_{R^i \in \mathcal{V}} \hat{A}_t^i$. Here, g is the total gradient that combines the gradients for all agents, with an expectation taken over the joint policy $\boldsymbol{\pi}$. Next, we rearrange the inner and outer sums, thus grouping the gradient with the corresponding advantage function. We then recognize the product of individual policies as the joint policy and lastly can simplify the expression by recognizing that the sum of the log-derivatives of individual policies is the log-derivative of the joint policy. This proof demonstrates that the cooperative policy gradient transforms into the gradient of the joint policy by summing the gradients of all agents' policies weighted by their respective advantages. This ensures that updates made according to this gradient ascent method will lead to the maximization of the expected cumulative rewards for the entire group of agents. Thus, the cooperative policy gradient aligns the individual agents' learning processes with the global objective of maximizing the joint reward, which is known to converge to a local maximum of the expected return $G_t = \sum_{R^i \in \mathcal{V}} G_t^i$ if (i) $\boldsymbol{\pi}$ is differentiable, (ii) the update timescales are sufficiently slow, and (iii) the advantage estimate uses a representation compatible with $\boldsymbol{\pi}$ [52]. □

[52]: Konda et al. (2000), *Actor-Critic Algorithms*

We detail the training algorithm in algorithm 1. Note that this blueprint can be applied to any policy gradient-based algorithm, such as PPO [55].

[55]: Schulman et al. (2017), *Proximal Policy Optimization Algorithms*

Algorithm 1: Cooperative policy gradient

Input: Initial policy parameters θ and value parameters used to estimate the advantage
for $k \leftarrow 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_t$ by running policies π_θ^i
  Compute advantage estimates $\hat{A}^i$ for all time steps
  Estimate the policy gradient as $g_k = \sum_{R^i \in \mathcal{V}} g_k^i$ with $g_k^i = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t \in \tau} \sum_{R^i \in \mathcal{V}} \nabla_\theta \log \pi_\theta^j(a_t^j | z_t^i) \hat{A}^i$
  Compute policy update $\theta \leftarrow \theta + \alpha_k g_k$
  Fit value functions
end

4.3 Experiments

We validate our proposed learning scheme and architecture on three case studies requiring communication between agents. For all our experiments, we use a PPO variation of our algorithm [55] and employ distributed training based on Ray [187] and RLlib [188]. We benchmark our approach against two random walk baselines. We emphasise that we consider the

[55]: Schulman et al. (2017), *Proximal Policy Optimization Algorithms*

[187]: Moritz et al. (2018), *Ray: A Distributed Framework for Emerging AI Applications*

[188]: Liang et al. (2018), *RLlib: Abstractions for Distributed Reinforcement Learning*

coverage problem as a specific case study for our training algorithm, and use it to show the intrinsic value of this algorithm (i.e. scalability and the ability to learn to communicate) rather than to improve on state-of-the-art, and therefore omit a comparison to first-principles-based coverage algorithms[189].

[189]: Galceran et al. (2013), *A survey on coverage path planning for robotics*

4.3.1 Environment setup

We evaluate our learning scheme in a custom grid world. Agents can communicate if they are closer than some predefined distance, thus defining the time-dependent graph $\mathcal{G}_t$. This graph changes as a function of the agents' positions and the communication range and is thus time-dependent. Each agent has a local field of view.

Each environment of size $W \in \mathbb{N}, H \in \mathbb{N}$ is initialized with a randomized obstacle layout and populated with N agents $\mathcal{V} = \{1, \dots, N\}$ placed at random initial locations. This randomness ensures generalisability to any unseen environment setup. The obstacles are placed while ensuring the environment remains connected, based on a random walk algorithm. Each agent R^i is described at discrete time t by its position $\mathbf{p}_t^i \in \mathbb{N}^2$ and binary occupancy map of the environment $\mathbf{M}_t^i \in \mathbb{B}^{W \times H}$ indicating positions of obstacles. Agents R^i and R^j can communicate to each other only if $\|\mathbf{p}_t^i - \mathbf{p}_t^j\| < d$, resulting in the communication graph $\mathcal{G}_t$. The communication range d is a hyperparameter of the environment.

At any time step, an agent can move in one of four directions or wait. The environment constrains actions to prohibit collisions with obstacles or the world's margin. Each agent R^i has a field of view (FOV) of width $W_{\text{FOV}} \in \mathbb{N}$ and height $H_{\text{FOV}} \in \mathbb{N}$. Each agent's partial observation can be described as tensor $z_t^i \in \mathbb{R}^{2 \times W_{FOV} \times H_{FOV}}$ and consists of channels for a local world map $\mathbf{M}_t^i$ (i.e., obstacle positions) and, depending on the experiment, an optional map $\mathbf{C}_t^i$ containing the agent's local information such as coverage or goal. We integrate the agent's position $\mathbf{p}_t^i$ implicitly into the observation by shifting and cropping $\mathbf{M}_t^i$ and $\mathbf{C}_t^i$ to the agent-relative field of view so that the agent is centred. We pad the observation so that the map outside the dimensions of $\mathbf{M}_t^i$ is perceived as occupied with obstacles and the optional map $\mathbf{C}_t^i$ with zeros (i.e., no coverage and no goals).

4.3.2 Non-convex coverage

The coverage path planning problem is related to the *covering salesman problem* in which an agent is required to visit all points in the target environment (an area or volume) as quickly as possible while avoiding obstacles [189]. Each agent keeps track of its local coverage $\mathbf{C}_t^i \in \mathbb{B}^{W \times H}$.

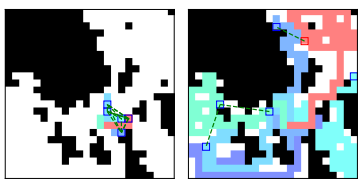


Figure 4.3: Standard coverage environment. Agents are visualised as blue and red squares. Black cells correspond to obstacles. Different colours indicate the coverage achieved by individual agents.

The global coverage $\mathbf{C}_t \in \mathbb{B}^{W \times H}$ is the logical disjunction $\mathbf{C}_t = \bigvee_{i=1}^{N} \mathbf{C}_t^i$ of all agents' coverages.

The grid-world has a size of $W = H = 24$ and is occupied with 40% obstacles that are randomly generated while assuring connectivity of the environment. Each agent receives a reward of 1 if it moves towards a cell that any agent has not yet covered. To incentivise fast coverage, in which all agents continuously cover new cells, and to increase sample efficiency, we use a dynamic episode length to terminate an episode if an agent has not covered a new cell for 10 time steps.

We train the cooperative policy for 20M training environment time steps. After completing the training, we evaluate with a fixed episode length $T = \lceil W \cdot H \cdot 0.6 \rceil = 346$, which is the minimum time required to visit every free cell for a single agent.

4.3.3 Split coverage

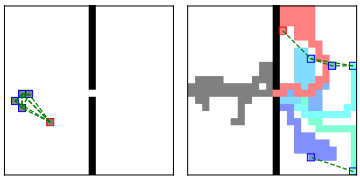


Figure 4.4: Split coverage environment. Agents are visualised as blue and red squares. Black cells correspond to obstacles. Different colours indicate the coverage achieved by individual agents.

The problem is similar to the non-convex coverage, but in this experiment, the agents only receive a reward for covering the *right-hand side* of the environment. Hence, the agents have to learn to first move to the right side and then coordinate to cover it.

The grid-world has a size of $W = H = 24$. The environment has a fixed layout split vertically into two halves by a line of obstacles. The two halves are connected through a single cell. We initialise the agent's positions similarly to the previous experiment but constrain them to be placed on the left side. The reward is similar to the non-convex coverage, but the agents are only rewarded for covering the right side of the environment. The early termination is similar but is only triggered if at least one agent has reached the right side. Otherwise, the episode ends after a fixed time.

We train the cooperative policy for 2M training environment time steps. We perform an evaluation with a fixed episode length $T = \lceil (W \cdot H)/2 \rceil = 288$, which is the time required to cover every cell in one half for a single agent.

4.3.4 Path planning

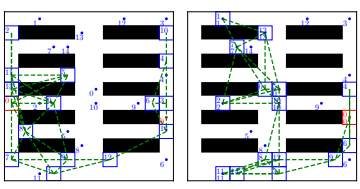


Figure 4.5: Path planning environment. Agents are visualised as blue and red squares. Black cells correspond to obstacles. Labelled goal locations are indicated by circles.

In the path planning scenario, each agent is required to navigate to a labelled goal $\mathbf{g}_t^i \in \mathbb{N}^2$. In contrast to the coverage scenarios, only one agent can occupy a single cell at the same time. Similarly to the coverage scenarios, each agent has access to its local occupancy map. Instead of the local coverage, each agent receives a relative grid map that indicates its respective goal location. Since each agent is only aware of the

relative environment layout and its own goal position, but not of other agent's positions, the agents have to communicate to coordinate and determine the most efficient set of paths. The communication channel and constraints are the same as in the coverage scenarios.

The grid-world has a size of $W = H = 12$. The environment resembles a warehouse layout with obstacles aligned in a regular grid. The agents are placed uniformly at random locations in the environment. Each agent receives a local reward of 1 at each time step t if $\mathbf{p}_t^i = \mathbf{g}_t^i$. Each episode has a horizon of $T = 50$ time steps. Therefore, the maximum possible mean reward per agent is T if each agent immediately reaches its goal position.

We train the policy for 16M training environment time steps. The episode length of T during evaluation is similar to during training.

4.4 Results

We compare the performance for three tasks with communication enabled and disabled (by setting an infinite communication range). We report qualitative results in Figure 4.3, Figure 4.4 and Figure 4.5, as well as quantitative results in Table 4.1, for which we average 100 evaluation runs for different environment seeds. We report the mean return with standard deviation.

4.4.1 Metrics

To quantitatively measure the performance for both coverage tasks, we create a more intuitive metric than local rewards for global coverage. Specifically, we introduce the function $p_{t,\pi}$ that returns the fraction of globally covered area for some policy π at time step t. This function considers only the area that can be covered (it excludes obstacles), and it is a global metric, meaning it will consider for all N agents if a cell has been covered or not.

Plotting $p_{t,\pi}$ over time allows a qualitative comparison of different policies. For example, a policy π_A is better than π_B if $p_{t,\pi_A} > p_{t,\pi_B} \forall t$, but it is theoretically possible that the performance curves of two policies intersect one or more times, which makes a quantitative comparison challenging.

In order to be able to quantify the performance, we calculate three distinct time steps $t_{0.5}$, $t_{1.0}$ and $t_{2.0}$ that represent the time at which an ideal policy π_{ideal} with N agents could reach either 50%, 100% or 200% coverage. We consider a policy ideal if each of the N agents covers a new cell at every time step until all coverable cells are covered.

Note that π_{ideal} is different from the optimal policy π_* defined in Section 2.5 since the ideal policy is not feasible in most cases. We refer to percentual coverage as $p_{t_{1.0},\pi} = p_{1.0}$.

For instance, in many non-convex environments, it is impossible to cover all cells without revisiting already covered cells.

4.4.2 Baseline

We introduce two baseline coverage algorithms we use for our performance evaluation: *1)* a random walk and *2)* a random walk with a heuristic. A randomised policy is straightforward to implement and a relevant baseline since it gives an intuitive understanding of what a lower performance bound looks like (analogous to the upper performance bound provided by π_{ideal}).

While related work on multi-agent coverage exists [189], we choose to focus on a random baseline to demonstrate the general feasibility of our learning strategy and highlight its ability to learn and scale as a proxy for a wide range of other relevant multi-robot problems.

[189]: Galceran et al. (2013), *A survey on coverage path planning for robotics*

The random walk policy π_r chooses one of the four actions to move up, down, left or right without considering the observation obtained from the environment. The second baseline policy $\pi_{r,h}$ adds a heuristic to the random walk that prefers moving neighbouring uncovered cells. Otherwise, a random action is chosen. We do not include any communication capabilities in our baselines. In all further evaluations, we use $\pi_{r,h}$ as a baseline. A sample visualisation of the performance metric over time for both baselines and the ideal policy can be seen in Figure 4.6.

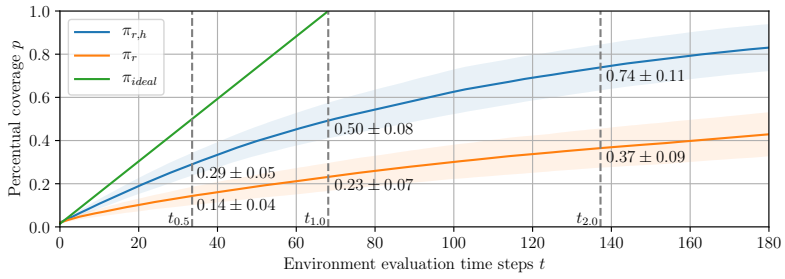


Figure 4.6: Example for ideal policy baseline used to generate a quantifiable performance metric at the example of two random baselines for a world of size $(24, 24)$ with $N = 5$ agents. The random baselines are plotted with a 1σ standard deviation (68% confidence interval). We plot the $p_{0.5}$, $p_{1.0}$ and $p_{2.0}$ metric next to the corresponding point in time at each graph.

4.4.3 Performance evaluation

In this section, we evaluate the performance for a variation of different model and environment hyperparameters. Specifically, we train all combinations of communication hops $K \in \{1, 3, 5\}$, communication range $d \in \{5, 10\}$ and number of agents $N \in \{3, 5, 10\}$, resulting in 18 trials. All trials train for 125M agent time steps, resulting in 25M environment time steps for $N = 5$, 41.6M for $N = 3$ and 12.5M for $N = 10$.

We first provide an overview of the three tasks before diving deeper into an analysis of the coverage task. The results indicate that the agents

are able to learn the respective tasks (for coverage, the agents learn to disperse while avoiding covering areas that have previously been covered by other agents, and for path planning, agents find their way to their respective goal locations). The qualitative results in Table 4.1 show that agents are able to improve their average performance by utilising explicit communication to coordinate.

Task	Communication w/	Communication w/o
Coverage	0.799 ± 0.042	0.643 ± 0.062
Split Coverage	0.529 ± 0.005	0.389 ± 0.11
Path planning	29.1 ± 6.0	22.4 ± 6.6

Table 4.1: Average return for all cooperative agents over 100 episodes at the end of training for all experiments given with a 1σ standard deviation. Rows show the tasks, and columns show the two communication variants.

Below, we report a detailed analysis of the coverage task. Provided are both performance plots for $N = 10$ in Figure 4.7 and the metric $p_{1.0}$ in Table 4.2. All measurements are accompanied by the 1σ standard deviation.

We proceed to analyse the performance for a policy trained with $N = 10$ and $d = 10$ and evaluated on $d = 5$ and $d = 10$ in Figure 4.7. First, we notice that there is little difference in performance for different values of K. $K = 5$ performs slightly better than $K = 3$, in particular around $t_{1.0}$. The standard deviation is slightly smaller for $K = 5$, indicating a more stable performance for different environment layouts. All trials outperform the random baseline, but the standard deviation is similar to it. For $d = 10$, we can see a noticeable increase in performance, but the standard deviation remains high. The reason for the poor performance with few agents and a low communication range is that a communication link between agents is rarely established, and therefore, no coordination occurs. This shows the working of the GNN.

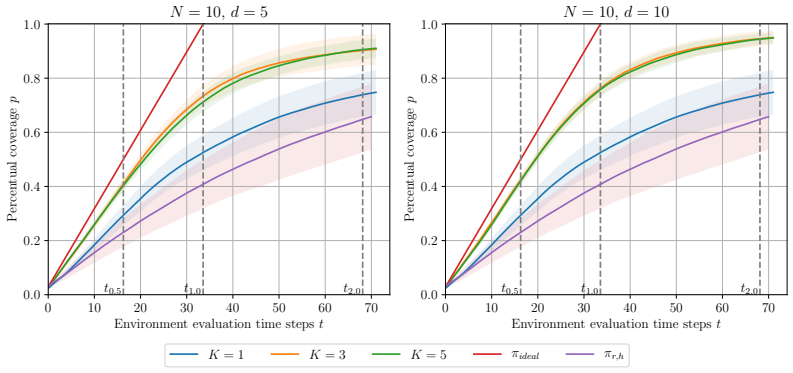


Figure 4.7: Performance plot for policy trained with $N = 10$ agents for communication distances $d = 5$ and $d = 10$.

We summarise all $p_{1.0}$ performances for all trials in Table 4.2. Since the performance metric captures the coverage speed and is normalised to the number of agents, a similar value for a different number of agents indicates that the approach scales well with the number of agents. Note that the performance is slightly lower for a larger number of agents. This is caused by the correlated initial placements of agents. With a

large number of agents placed in a similar position, some agents have to move along already-covered cells to find uncovered cells. This effect becomes more noticeable with a larger number of agents. Therefore, this approach, at least with the correlated initial placement of agents, never scales linearly.

Table 4.2: Summary of the 100% performance $p_{1.0}$ for all combinations of trials. We can see $d = 10$ outperforms $d = 5$ and that $K = 3$ outperforms $K = 5$.

	$d = 5$			$d = 10$			
	$K = 1$	$K = 3$	$K = 5$	$K = 1$	$K = 3$	$K = 5$	Random
$N = 3$	0.670 ±0.074	0.698 ±0.080	0.703 ±0.080	0.670 ±0.074	0.736 ±0.072	0.756 ±0.067	0.542 ±0.088
$N = 5$	0.643 ±0.062	0.722 ±0.055	0.700 ±0.040	0.643 ±0.062	0.788 ±0.042	0.759 ±0.034	0.488 ±0.101
$N = 10$	0.530 ±0.066	0.739 ±0.039	0.717 ±0.037	0.530 ±0.066	0.768 ±0.029	0.763 ±0.026	0.412 ±0.090

Figure 4.8 visualises the fraction of $p_{1.0}$ for the trained policy and the random policy (*e.g.*, a value of 1 indicates that the trained policy has covered the same area as $\pi_{r,h}$ at $t_{1.0}$). This plot can be interpreted as a metric of how much better a policy for N agents performs than the corresponding random heuristic $\pi_{r,h}$. We can see an almost linear increase for $K = 3$ and $K = 5$, while there are no changes for $K = 1$, which is to be expected. This indicates that the trained policy scales much better with increasing the number of agents than the random policy.

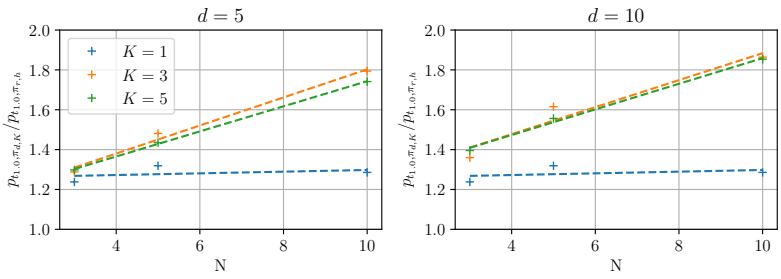


Figure 4.8: We compute the fraction of the coverage for the trained policies for different K over different numbers of agents N at any time t. Expectedly, there is no speedup for $K = 1$ (no communication) but a noticeable speedup with communication for $K > 1$, which indicates that the communication-based policies scale better with the number of agents than the random policy.

Lastly, we evaluate the coverage at $t_{1.0}$ for different environment samples in Figure 4.9. We observe that the agents are capable of distributing in the environment and manage to avoid each other. There is some overlap of less than two cells for some neighbouring coverages, which results in incomplete coverages for other parts of the field (white). Environments consisting of mostly free spaces reach higher coverage (experiments 2, 3, 11, 16, 20) than cluttered environments (experiments 7, 8, 15).

4.4.4 Generalization

Next, we evaluate the generalisation performance of a different number of agents during training at evaluation time. We choose $K = 3$, $d = 10$ as best policy and $N \in \{3, 5, 10, 15\}$ for all generalization trials. After training, we attempt to generalise during inference to N_g (the number of agents with which to run the evaluation).

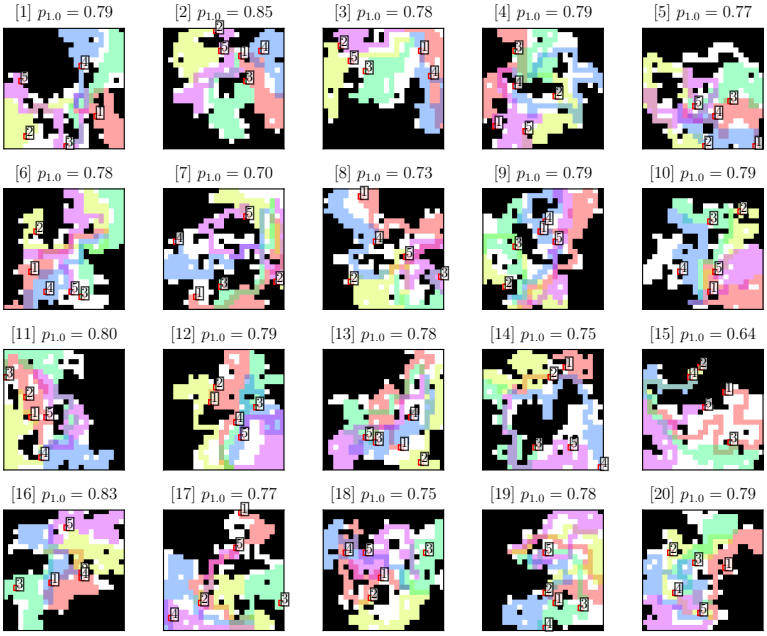


Figure 4.9: Coverage for different environments at $t_{1.0}$. The individual coverages of the $N = 5$ agents are highlighted in different colours. The agent's positions are highlighted as numbered red squares.

Table 4.3 shows an overview of performance plots for the generalisation of N to $N_g \in \{3, 5, 10, 15\}$, color-mapped from green (best) to red (worst) to highlight performance differences. All trials indicate the generalisation capabilities of the trained policies with a degraded performance if trained with a large number of agents and evaluated with few agents (and vice versa). Generally, the policies perform best if N and N_g are close to each other.

The performance degradation for $N_g > N$ can be partially explained with a general performance degradation for a larger number of agents. Due to the initial placement of agents and the necessity for more agents to move along already visited tiles, if agents are placed close to each other, more agents are forced to move along already covered cells. This is a flaw in the performance metric that assumes that the ideal policy π_{ideal} scales with the number of agents.

We also notice some performance degradation for $N_g < N$. While $p_{1.0}$ is 0.746 for $N = N_g = 15$ and 0.735 for $N_g = N = 3$, it is only 0.496 for $N = 15$, $N_g = 3$. The policy generalises worse for $N_g < N$ than for $N_g > N$.

$N_g =$	3	5	10	15	20	25	30
$N = 3$	0.735 ±0.071	0.787 ±0.039	0.726 ±0.033	0.654 ±0.029	0.558 ±0.024	0.437 ±0.019	0.386 ±0.017
$N = 5$	0.707 ±0.061	0.785 ±0.035	0.771 ±0.027	0.718 ±0.024	0.620 ±0.019	0.500 ±0.015	0.443 ±0.012
$N = 10$	0.573 ±0.059	0.715 ±0.045	0.776 ±0.026	0.745 ±0.026	0.642 ±0.022	0.531 ±0.014	0.468 ±0.011
$N = 15$	0.496 ±0.070	0.677 ±0.046	0.761 ±0.030	0.746 ±0.024	0.640 ±0.019	0.528 ±0.016	0.468 ±0.011

Table 4.3: Performance $p_{1.0}$ for a all generalization trials.

A possible explanation for the lower performance, particularly for $N_g < N$, is that the trained policy and, therefore, the optimal behaviour differs for a different number of agents. Another reason is that the three policies for $N \in \{3, 5, 10, 15\}$ agents were trained for the same number of individual agents' time steps but for a different number of global time steps. It is possible that policies with more agents generally require longer training to achieve the same performance.

4.4.5 Policy interpretation

Understanding and interpreting a NN is essential to verify its functionality and to improve shortcomings. Since NNs are generally black boxes, and we train our solution end-to-end, understanding the inner workings is particularly challenging. In this chapter, we interpret a trained policy both with guided backpropagation and with a supervised learner by reconstructing global and the agent's local coverages.

Guided backpropagation

Guided backpropagation is a method to visualise the gradient in a NN with respect to an individual neuron conditioned on an input image and was introduced in [20]. This method is a combination of the deconvolutional network (deconvnet) approach [190] and a plain backpropagation after a forward pass. The idea is to inverse the dataflow in a CNN. Starting at a feature map obtained after a single forward pass of the NN, we would like to know what parts of the input data contribute most to the activation of an individual neuron. All other outputs in the feature map except the desired neuron are set to 0. After performing a backward pass with this modified feature map, a reconstruction of the input data is generated, indicating the section of the input data that most activated the given neuron. This method is restricted to the ReLU activation [28], which applications mask out certain values in the forward and backward pass and require separate handling for other activation functions. Guided backpropagation combines a plain backpropagation with the deconvnet approach and masks out values corresponding to negative entries in both directions of the gradient. This adds an additional guidance signal by preventing the backward flow of negative gradients (neurons that decrease the activations to visualise) that is particularly helpful when visualising features from higher levels of a CNN.

[20]: Springenberg et al. (2015), *Striving for Simplicity: The All Convolutional Net*

[190]: Zeiler et al. (2014), *Visualizing and Understanding Convolutional Networks*

[28]: Nair et al. (2010), *Rectified linear units improve restricted boltzmann machines*

We generalised the implementation of FlashTorch [191] to handle the more complex and nonlinear computation graph used in this work. After performing the forward pass, which computes the action distribution, we perform guided backpropagation on the action logit that corresponds to the chosen action. This way, given the action performed by an individual

[191]: Ogura et al. (2020), *FlashTorch: Visualization toolkit for neural networks in PyTorch*

agent, we can generate a set of tensors of the same shape as z_t^i (the observation) for all agents R^i, indicating how much the observations of all agents contributed to the predicted action.

With this experiment, we aim to understand why a given agent chose an action. We choose the policy trained with $K = 3$ and $d = 10$ as the best-performing policy from the trials in Section 4.4.3.

Figure 4.10 shows the execution of a trained policy at four distinct time steps with guided backpropagation performed on the neuron corresponding to the chosen action of agent 1, so the backpropagation heatmaps show what part of all observations of all agents contributed most to choosing that action.

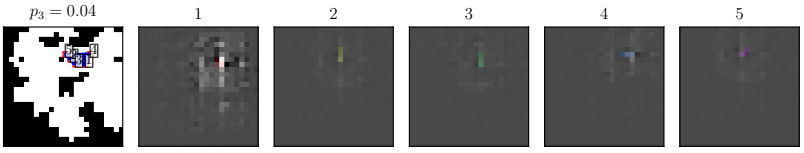
Figure 4.10a shows the initial dispersion of the agents from their initial location. In this phase, the communication graph is fully connected since all agents are closer than the communication range d. Communication is particularly important in this early phase since an optimal policy would aim to cover a new cell at every step of the process for every agent. If the agents do not disperse properly, they will have to move along cells already covered by other agents. For a larger number of agents, this may be unavoidable in the beginning. During the initial dispersion, it is also decided what sub-area of the environment is covered by which agent, so this initial phase is critical for the final performance.

Figure 4.10b shows the second phase, which mostly focuses on completing the individual coverage in the area the agent moved toward. During this phase, the guided backpropagation mostly highlights the agent's local coverage. This is also caused by the agents being frequently out of communication range. This indicates a good generalisation of this use case and high robustness against loss of communication of the policy. If the agent moves close to neighbouring agents and re-gains communication, as can be seen in Figure 4.10c, the neighbour's local coverage is taken into consideration, and an action is chosen that avoids moving into the neighbour's area.

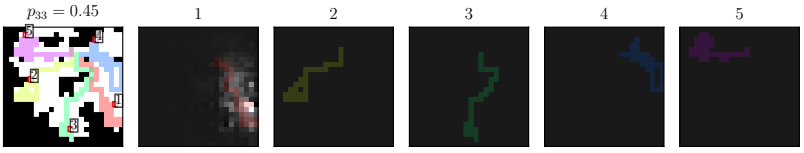
We run the policy past $t_{1.0}$ (which is 70 in this setup) to observe how the coverage continues past this point. We noticed that the magnitude of the activations increased and that, generally, most agents contribute to the chosen action as long as the agents are connected. The general activation across the team seems to increase with the coverage.

Supervised interpreter

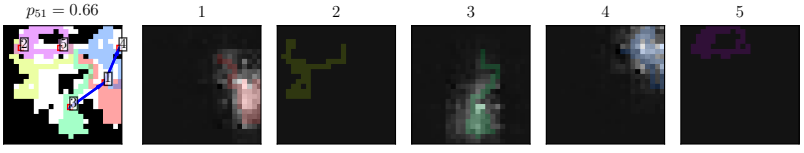
Our hypothesis of the role the GNN plays in the coverage is that it contains an agent-relative representation of the global coverage $\mathbf{C}_t$ that is aggregated over multiple communication hops K based on all agent's CNN output $\mathbf{h}_{0,t}^i$. The aggregated global coverage can then be used by



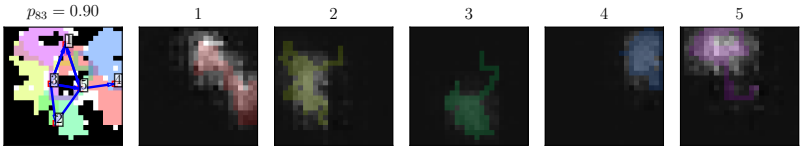
(a) $t = 3$: Dispersion of the agents from their initial placement in the random environment.



(b) $t = 33$: Agent acting exclusively on the local coverage map and not using any communicated data.



(c) $t = 51$: High activity for local coverage of agent 3 (immediate neighbour) and proof of multi-hop communication to agent 4 (low activity in the rightmost map)



(d) $t = 83$: High (multi-hop) activity from all agents after the coverage is almost completed.

Figure 4.10: We execute a trained policy with $N = 5$ agents and visualise the activation of the output neuron of the last chosen action for agent 1. The leftmost panel summarises the world state. The agents are numbered and assigned an individual colour for the individual coverage. The arrows indicate the $K = 3$ communication range of agent 1. The five panels on the right show the local coverage in the same colour for the corresponding agents together with the output of the guided backpropagation (white means highest activation, black means lowest activation). $t_{1.0} = 70$ in this environment setup.

the post-processing MLP to predict an action that helps the team to complete the global coverage. This is in line with the capabilities of the homogenous communicating agents defined in Section 2.1.2.

We test this hypothesis by training a supervised interpreter f_ψ^{-1} that minimises the reconstruction error such that

$$f_\psi^{-1}(\mathbf{h}_{K,t}^i) \approx \mathbf{C}_t \; \forall \, (\mathbf{C}_t, \mathbf{h}_{K,t}^i) \in \mathcal{D}_{\text{interpreter}}$$

The function f_ψ^{-1}, represented by a CNN with weights ψ, maps each agent's GNN output feature vector $\mathbf{h}_{K,t}^i$ to the agent observation z_t^i.

We collect 50K training, 10K validation and 5K testing samples $\mathcal{D}_{\text{interpreter}}$ consisting of tuples $(\mathbf{C}_t, \mathbf{h}_{K,t}^i)$ that are obtained by evaluating the policy π trained with $K = 3$ and $d = 10$. To reduce the correlation between samples, we sample at every time step t with a probability of 10% and discard all other samples.

The neural network architecture f_ψ^{-1} consists of multiple combinations of 2D convolutions, LeakyReLU activation, 2-upsampling and zero padding where the specific hyperparameters depend on the input feature size and the desired output size. The final convolution has one output channel and a sigmoid activation. We use a binary cross-entropy loss. We are only interested in the features that are essential for an efficient collaboration

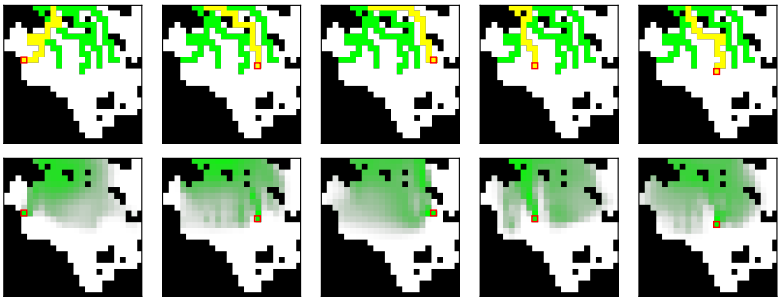
between agents and, therefore, mask the loss and evaluation metrics correspondingly.

We train with a batch size of 64 for 200 training epochs. Every ten training epochs, we evaluate the classification performance as mean average precision (mAP) on the validation set. We avoid overfitting by check-pointing the model with the best mAP computed on the validation set. All interpreter visualisations in this chapter are created from a subset of the testing samples.

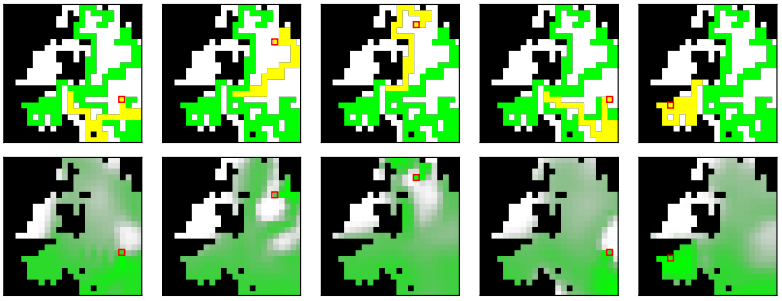
We train the interpreter to reconstruct the global coverage for every agent from the concatenation of the own CNN output that captures and compresses the own observation and the output of the GNN that receives and processes information from the other agents. We expect to be able to reconstruct the true global coverage of the environment as the GNN should aggregate this information. Three samples are visualised in Figure 4.11.

Figure 4.11a shows the reconstruction during an early phase. The area that is not covered by any agents is predicted correctly, but the areas agents actually visited are slightly blurry. We observe that the prediction of global coverage is more accurate in the immediate vicinity of each agent. This makes sense since the the environment closely around the agent is critical to determine the action to move to one of the neighbouring cells, but for cells further away only a broad sense of direction is sufficient.

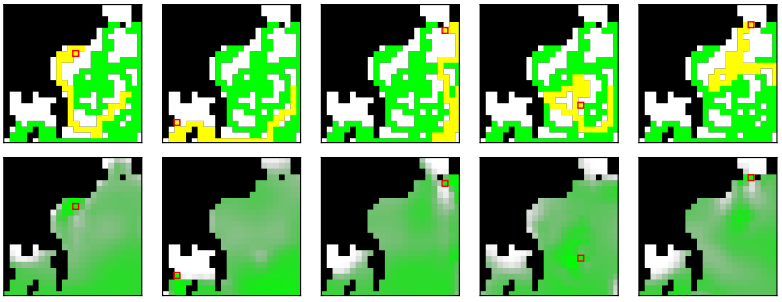
Figure 4.11b and Figure 4.11c show two more samples at different coverages. We can see that the areas covered by the corresponding agent itself (yellow) are reconstructed more clearly from that agent's CNN feature vector. The other parts of the area are reconstructed from the GNN feature vector and are a bit more blurry. Even though they differ slightly from agent to agent (which is to be expected due to different agents that can be reached with K graph hops), the general shape of the coverage is nicely reconstructed. The prediction is particularly clear for larger areas without coverage. Such areas are more relevant since they potentially result in a higher reward since more cells can be covered.



(a) Top: ground truth, bottom: prediction.



(b) Top: ground truth, bottom: prediction.



(c) Top: ground truth, bottom: prediction.

Figure 4.11: Example of reconstructed global coverage with the help of a supervised interpreter. The first row in each subfigure shows the ground truth for all $N = 5$ agents at the same time step. The individual coverage $\mathbf{C}_i$ of each agent is highlighted in yellow, and the coverage of the other agents is highlighted in green. The yellow coverage is accessible through each agent's own observation, but the green coverage can only be reconstructed from the global coverage aggregated through the GNN.

4.5 Conclusion

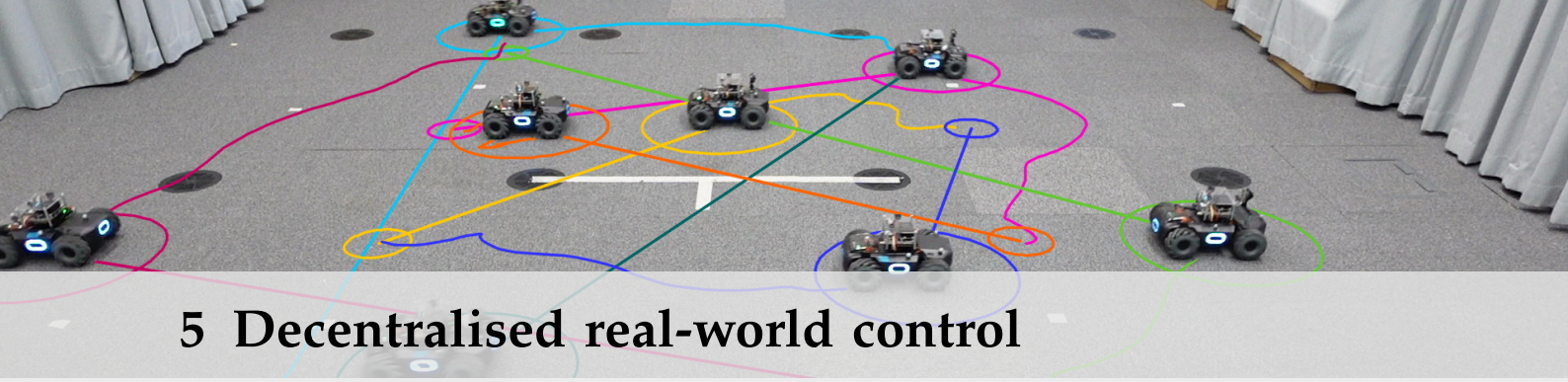
We introduce a novel MARL training model and algorithm for learning to communicate in MAS with local agent-specific rewards while maintaining a common trainable communication channel.

We used our method to train the multi-agent sweep coverage or CPP problem in a simulated grid-world. We evaluated the implemented solution in terms of scalability to a larger number of agents and generalizability to a different number of agents than it was trained for. We found that the solution scales better than a random baseline to a larger number of agents, and the multi-hop information sharing significantly benefits the coverage problem. We then applied different explainability methods to our solution. We used guided backpropagation to understand the contribution of the coverage of other agents in the communication graph to the decision to choose a certain action. Our hypothesis of the functionality

of the GNN in our application is that it builds an estimate of the global coverage that individual agents can use to decide where to move at any time. We test this hypothesis by sampling the GNN output of a trained policy together with the ground truth coverage and reconstructing the global coverage from this latent space with supervised learning.

This chapter is the foundation for an extension that was not further discussed as part of this thesis, in which we aim to demonstrate the malicious behaviour of a competitive group of agents. We instantiate a greedy agent with a group of cooperative agents in the same environment, competing for the same resources (the coverage). Both groups of agents share their feature vectors but train separate policies. Since the greedy agent integrates the observations of the cooperative group into its GNN, it can benefit from the shared coverage information of that group. Similarly, the competitive group will integrate information shared from the greedy agent into its GNN. Since the greedy agent is competing on its own, malicious behaviour emerges (the greedy agent sends information that corrupts the functionality of the GNN of the cooperative group). Due to these manipulations, the malicious agent claims more coverage than it could without manipulating the communication. In order to prove the malicious behaviour, it was essential to understand the problem and the solution well, which is the primary reason for developing extensive explainability tools.

The work in this chapter is the foundation for real-world deployments of MRS. A major limitation is the reliance on ground-truth state information, such as relative pose, and maintaining local coverage maps. Furthermore, discrete actions are not reflective of the real world. We address these limitations in the next chapter.



5 Decentralised real-world control

In the previous section, we set the foundations for training decentralised cooperative multi-robot coordination policies with RL that use GNNs at their core. In this section, we focus on the real-world deployment of our policies to a decentralised MRS.

While GNN-based policies are typically trained in a centralised manner in simulation and assume synchronous communication, resulting policies can be executed either in a centralised or decentralised mode. Evaluating a GNN in the *centralised* mode typically requires execution on a single machine decoupled from the robots acting according to the policy. 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 *decentralised* mode, each robot is responsible for making its own decisions. With decentralised evaluation, *(i)* there is no single point of failure, resulting in higher fault tolerance, *(ii)* agents do not need to remain in the network range of a router that orchestrates the evaluation, and *(iii)* computation is parallelised across N robots, decoupling the time complexity from the number of robots.

Even though GNNs have an inherently decentralisable mathematical formulation, previous work on GNN-based multi-robot coordination was conducted exclusively in centralised simulations using synchronous communication [3, 90, 192]. For practical reasons, decentralised 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 [193–196]. Further compounding these issues, decentralised policies are typically executed asynchronously, resulting in system states not previously encountered during training.

In this chapter, we address these challenges by first introducing a new multi-robot research platform and framework that facilitates the decentralised execution of GNN-based multi-robot policies. We use this platform for experiments throughout the remainder of this thesis. We

5.1 Contributions 62
5.2 Approach 62
5.2.1 Cambridge RoboMaster . 63
5.2.2 Deployment taxonomy . . 64
5.2.3 ROS2 deployment 65
5.3 Experiments 67
5.3.1 Communication infrastructure 68
5.3.2 Navigation in cluttered environments 69
5.4 Results 73
5.4.1 Networking 73
5.4.2 Narrow constriction 74
5.4.3 Scaling to eight robots in VMAS 76
5.5 Conclusion 78

[3]: Li et al. (2020), *Graph neural networks for decentralized multi-robot path planning*
[90]: Tolstaya et al. (2020), *Learning Decentralized Controllers for Robot Swarms with Graph Neural Networks*
[192]: Kortvelesy et al. (2021), *ModGNN: Expert Policy Approximation in Multi-Agent Systems with a Modular Graph Neural Network Architecture*
[193]: Calvo-Fullana et al. (2021), *Rosnetsim: A framework for the integration of robotic and network simulators*
[194]: Gregory et al. (2016), *Enabling autonomous information-gathering and self-recovery behaviors in partially-known, communication-constrained environments*
[195]: Stephan et al. (2017), *Concurrent control of mobility and communication in multirobot systems*
[196]: Stephan et al. (2014), *Robust routing and multi-confirmation transmission protocol for connectivity management of mobile robotic teams*

proceed to introduce a taxonomy of different evaluation modes and networking configurations, as well as four different modes of deployment, ranging from centralised to decentralised. We then present the results of a suite of real-robot experiments with up to five robots to demonstrate the consequences of this decentralised execution. We include additional centralised experiments showcasing the deployment of up to eight robots with emerging behaviour.

5.1 Contributions

Our core contributions for this chapter are:

- An omnidirectional indoor ground robot based on the DJI Robomaster S1, which is *affordable* (starting at USD 689 in the base configuration to USD 1633 with all sensors and a Jetson Orin NX), *agile* (we demonstrate indoor trajectory tracking capabilities of up to 4.45 m/s and accelerations of up to 5 m/s^2), *versatile* (we show experiments in classical control to zero-shot deployment of MARL policies trained in our simulator VMAS [197]), and is easily *replicable* (our entire stack is open-sourced, requiring minimal construction time). We position our platform with respect to available research platforms in Section 3.3. Our solution sits at a frontier beyond which agility (speed) is attained only by aerial platforms, and a lower cost is only attained by sacrificing agility.
- A Robot operation system 2 (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 centralised or decentralised manner. We provide the source code online.
- An ablation study on several forms of execution to quantify performance shifts between centralised execution and three forms of decentralised policy execution, *(i)* offboard (non-local), *(ii)* onboard over routing infrastructure, and *(iii)* onboard with ad-hoc networking.

Supplementary material

We provide the following supplementary material:

Code: Video:

And the following supplementary material specifically for the Cambridge RoboMaster:

Website: Code:

[197]: Bettini et al. (2022), *VMAS: A Vectorized Multi-Agent Simulator for Collective Robot Learning*

5.2 Approach

In this section, we detail our custom multi-robot research platform, the Cambridge RoboMaster, as well as our deployment taxonomy, ranging from centralized to decentralized, and our ROS2 deployment infrastructure.

5.2.1 Cambridge RoboMaster

In this section, we introduce the Cambridge RoboMaster, a new multi-robot research platform, and infrastructure that helped us in the deployment of a fleet of up to eight robots, as well as a framework that facilitates the decentralised execution of GNN-based multi-robot policies.

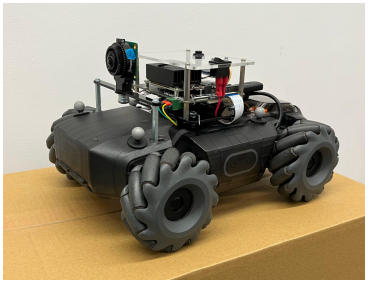


Figure 5.1: We show a close-up of one RoboMaster equipped with an NVidia Jetson Orin NX and a forward-facing camera.

Deploying MRS necessitates robotic platforms that can be used as testbeds for rapid development and evaluations. The choices for such platforms are usually informed by several factors, such as the problem's configuration space, the number of agents involved, and the computational sophistication expected from each agent. For instance, small-scale robots, such as the Turtlebot [134], will frequently be limited in their agility of motion and are restricted in their onboard computing capabilities. On the other hand, while larger robots (Jackal [116] or AutoRally [100]) have greater manoeuvrability and can carry more compute payloads, operating multiple of them in smaller research spaces can be cost- and space-prohibitive. Successfully testing and developing multi-robot solutions requires platforms that strike a fine balance between these objectives. This section presents an agile and affordable system and platform, shown in Figure 5.1 as a multi-robot team setup. Our system, an omnidirectional indoor ground robot based on the DJI Robomaster S1, is affordable, agile, and easily replicable. These were made possible by our retrofitting of the onboard compute module and careful reverse engineering of the main communication interface. This section details the background and design decisions made when developing our robot platform. In this section, we introduce the platform and the reverse-engineering efforts necessary to make it work, as well as the infrastructure necessary to deploy and control the MRS.

[134]: Turtlebot3 (2024), *TurtleBot3 Website*

[116]: Clearpath Robotics Inc. (2020), *JACKAL Unmanned Ground Vehicle: Technical Specifications*

[100]: Williams et al. (2016), *Aggressive driving with model predictive path integral control*

Platform We selected the DJI RoboMaster S1 as our platform's base, leveraging its design inspired by the RoboMaster competition—a DJI initiative to foster technological exchange and innovation among regional universities [198]. The S1, an educational robot designed for indoor use, integrates a tank-like structure with a spring-dampened beam axle front suspension and a four-wheel omnidirectional drive system. It features a gimbal-mounted blaster toy gun and an onboard computer. DJI specifies the base platform's maximum speeds as 3.5 m/s forward, 2.5 m/s backward, 2.8 m/s laterally, and 600 deg/s rotationally, with a weight of 3.3 kg (note that as part of our contribution, we side-step these manufacturer-imposed restrictions). Each wheel has a diameter of 100 mm, is powered by a brushless motor with integrated gears and a driver and supports closed-loop speed control up to 1000 RPM and a maximum torque of 0.25 Nm. This gives the platform a theoretical maximum speed of $(\pi \cdot 0.1\,\text{m} \cdot 1000\,\text{RPM})/(60\,\text{s/M}) = 5.23\,\text{m/s}$. The

[198]: Committee (RMOC) (2022), *RoboMaster 2023 University League Rules Manual*

system is powered by a 25.91 Wh smart battery, offering 2.4 Ah capacity. [199].

[199]: DJI RoboMaster S1 (2024), *DJI RoboMaster S1 Website*

The onboard computer interfaces with a sub-controller, managing all sensors and wheels. We replaced the onboard computer, gimbal, and blaster gun with a custom computing solution. We laser-cut components from acrylic to mount our custom expansion in place of the turret. All our design files are also part of our open-source release. [1]

1: Code:

Compute The stock onboard compute module on the RoboMasters is impractical for running custom algorithms or implementing additional interfaces to cameras, sensors, networks, etc. We, therefore, upgraded the onboard computer to a more powerful and versatile single-board-computer (SBC), focusing on the NVidia Jetson Orin family for its reliability and lifecycle availability until January 2030. While the Jetson Orin NX 16GB was selected for its robust processing capabilities and suitability for our research, our design easily admits other cost-effective single-board-computers such as the Raspberry Pi. The Jetson Orin series necessitates a carrier board. Our choice, the AverMedia D131 carrier board, was influenced by the availability of a Controller area network (CAN) interface – essential for communicating with the RoboMaster sub-controller – and an optimal balance of interface availability and cost. It accommodates two PCIe M.2 slots, where we installed a 256 GB SSD and an Intel 9260 WiFi module. In an alternative Raspberry Pi setup, we used the built-in ports and interfaces and added an additional CAN shield for communication with the RoboMaster base. We provide additional details on the components, the reverse engineering, and the deployment infrastructure in Section A.1 and Section A.2.

5.2.2 Deployment taxonomy

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 decentralised asynchronous ad-hoc mode, but for the purpose of an experimental ablation analysis, we identify a range of sub-categories with different degrees of decentralisation.

Specifically, we introduce the four modes: Centralised (fully centralised evaluation), Offboard (asynchronous evaluation on a central computer), Onboard o/Infra (decentralisation using existing centralised networking infrastructure) and Onboard o/Ad-hoc (full decentralisation using ad-hoc communication networks), as visualised in Figure 5.2. We describe the networking considerations that allow ROS2 to be used for decentralised ad-hoc communication between agents.

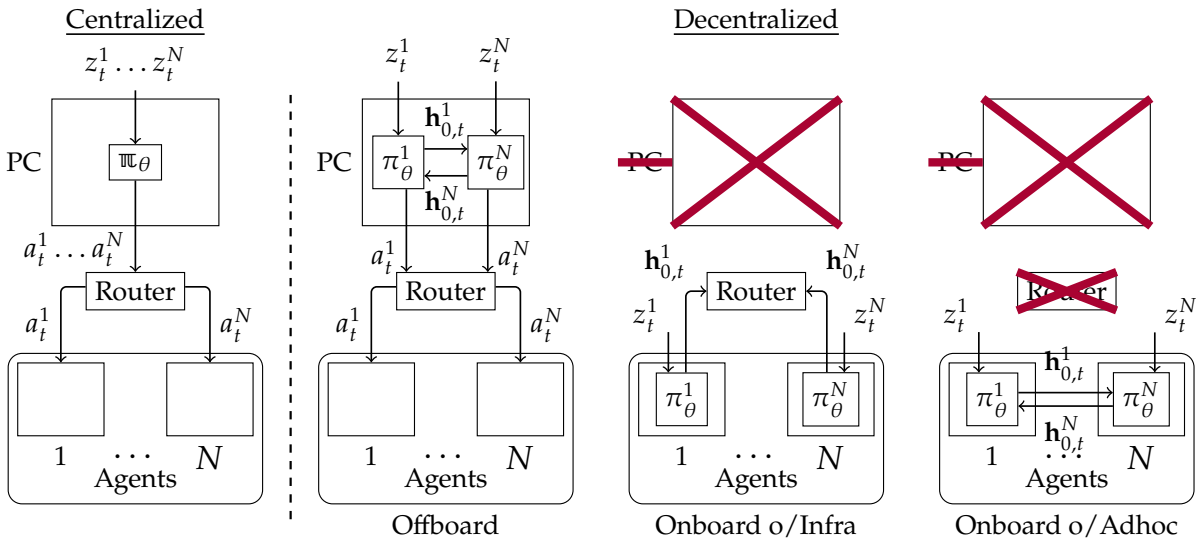


Figure 5.2: The framework configurations used in our experiments. We refer to these four configurations as Centralised, Offboard, Onboard over Infrastructure, and Onboard over Adhoc. The ROS2 infrastructure is either centralised or decentralised, with varying degrees of decentralisation depending on the network setup. Observations $z_t^1 \dots z_t^N$ feed into centralised policy $\boldsymbol{\pi}_\theta$ or local policies $\pi_\theta^1 \dots \pi_\theta^N$ to produce actions $a_t^1 \dots a_t^N$ for agents $1 \dots N$. Local policies consist of a GNN and pass messages $\mathbf{h}_{0,t}^1 \dots \mathbf{h}_{0,t}^N$ to communicate. In the centralised case, a single policy produces actions for all agents at once in a synchronised 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/Ad-hoc is fully decentralised – the agents forgo the router and communicate directly using ad-hoc networking.

5.2.3 ROS2 deployment

Robot operation system (ROS) is a set of open-source libraries for messaging, device abstraction, and hardware control [200]. 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 decentralised systems. ROS2 is a redesign of ROS that solves the master node issue, enabling completely decentralised systems [201]. Many popular frameworks have not migrated from ROS to ROS2, preventing their use in decentralised MRS. Our software infrastructure leverages ROS2 to create fully independent agents.

[200]: Quigley et al. (2009), *ROS: an open-source Robot Operating System*

[201]: Open Robotics (2021), *ROS 2 Design*

Our evaluations consider four different configurations, as summarised in Figure 5.2, 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. Centralised 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 for various configurations. Both use 802.11, with the first being an *Infrastructure mode* network, and the second being *ad-hoc mode*. We selected 802.11 in preference to other ad-hoc capable wireless standards, such as IEEE 802.15, due to achievable data rates and compatibility with IP-based networking.

Infrastructure mode This mode is characterised by a central access point being responsible for managing the network's functions. For Centralised 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, the Onboard o/Ad-hoc configuration handles only the delivery of agent location observations. The implications of each of these modes are discussed further in Section 5.4.1.

Ad-hoc mode We use this network mode only in the Onboard o/Ad-hoc mode, which handles messages between agents. Physically, this network is supported by distinct wireless transceivers carried with each agent, allowing decentralised 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-Data Distribution Service (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.

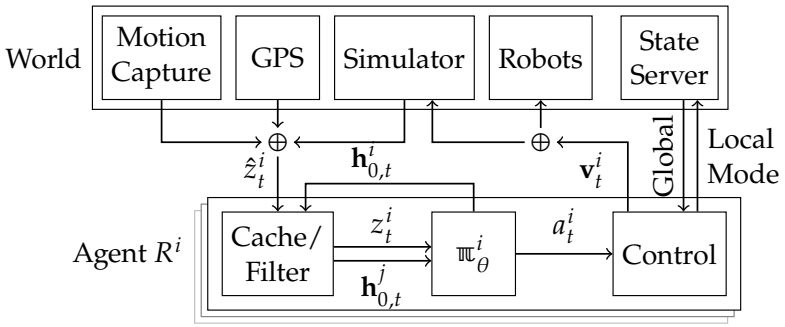


Figure 5.3: Our ROS2 architecture is composed of the *world* and *agents*. The agents receive sensor information from the world. The aggregator combines sensor information with messages to produce observation and neighbourhood messages for the policy to predict an action. The control node converts the action into velocity commands. The state server orchestrates termination, resets, and operational mode syncs during sequential episodes. This system allows us to run agents over multiple episodes without any human intervention.

Our multi-agent ROS2 architecture (see Figure 5.3) allows us to run both simulated and real-world agents concurrently, over multiple episodes, in centralised or decentralised mode, and without human intervention (facilitated through automated resets) and is composed of the *world* and *agents*. There is one agent $R^i = 1$ in the centralised case and multiple $R^i \in \{1 \dots n\}$ in the decentralised case. The agents receive sensor information $\hat{z}_t$ from either the motion capture system, GPS, or simulator. The aggregator combines sensor information with messages $\mathbf{h}_{0,t}^i$ to produce observation z_t^i and neighborhood messages $\mathbf{h}_{0,t}^j; R^j \in \mathcal{N}_t^i$, for the policy π_θ^i to generate action 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. 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 RL paradigm of delineating the *agent* from the *world*. This system allows us to run agents in simulation and the real-world concurrently, over multiple episodes, and without any human intervention.

Agent The 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 $\hat{z}_t^i$ to determine neighbouring agents $R^j \in \mathcal{N}_t^i$ within the specified communication radius. It caches neighbourhood messages $\mathbf{h}_{0,t}^j$ and sensor information z_t^i over Δt for the policy. The policy node wraps a trained policy π_θ^i. It receives the observation z_t^i and messages $\mathbf{h}_{0,t}^j$ and emits a message $\mathbf{h}_{0,t}^i$ and action 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 and then broadcasts a global operating mode and initial conditions. Agents use the global operating mode to determine if they should reset or execute the policy.

5.3 Experiments

In this section, we describe the experimental setup. We conduct experiments for the peer-to-peer communication infrastructure and describe the training setup for our case study, the navigation of an MRS through a narrow constriction.

5.3.1 Communication infrastructure

We investigate the effects of networking, ROS2, and Fast DDS settings on performance. During evaluations, our primary metric is the probability of delays between packet transmission and reception for message transmission rates between 10 and 500 per second, where the optimal case is all messages delivered with no delay (approximated in the Centralised and Offboard configurations).

We carry out all network-specific experiments on a platform of five Raspberry Pis spaced 2m to 10m away from each other in the same lab environment described in Section 5.3.2.

Multicast vs Unicast We use the eProsima DDS RTPS implementation [202], which defaults to using unicast (one-to-one) communications between publishers and subscribers. This allows reliable transport protocols; however, when multiple subscribers are active, the publisher will send duplicate messages as many times as there are subscribers, leading to exponential increases in messaging rates with increasing agent counts. The alternative is to use Multicast, where each publisher sends only one wireless broadcast for each message. The drawback is that neither reliable transport protocols nor the 802.11 hardware-based re-transmission mechanism can be utilised, reducing the odds of a given message being delivered.

[202]: eProsima (2020), *Fast-DDS*

802.11 hardware retries When using 802.11, as the number of competing nodes goes up, the probability of any given packet surviving transmission goes down. This is because if two or more nodes transmit at the same time, both packets are lost, and there is no coordination mechanism. For unicast messages, the lack of an acknowledgement from the receiver will cause re-transmission attempts up to a limit. This limit defaults to 7, and we evaluate the performance of 1, 3, 5 and 7 in our testing. We focus on lower settings than default because these reduce contention.

Wireless adapter selection and channel bandwidth The data rate of the network is dependent upon the distance between participants, transmit power, receiver sensitivity, 802.11 version and channel bandwidth. The final configuration used the Netgear A6210 adapter, based upon the MediaTek MT7612U chipset, and a 40MHz channel width. The adapter was selected because it runs a recent 802.11 version (802.11ac) and has Linux driver compatibility with IBSS mode. This adapter has a maximum transmit power of 18dbm; this is sufficient for ranges as high as 20m between robots even in the presence of interference from neighbouring 802.11 traffic, though communications over 200m were possible in quiet environments.

Reliability When RTPS-based reliability is enabled through the ROS2 configuration, subscribers notify publishers when messages are not received as expected through different mechanisms. One of these is the use of positive acknowledgements by subscribers, which allows publishers to re-transmit when messages are lost but causes subscribers to generate additional packets.

5.3.2 Navigation in cluttered environments

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 Figure 5.4. 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 neighbours to find the fastest collision-free trajectory to its respective goal. An image of this setup can be seen in Figure 5.4.We briefly explain the training and provide the code with implementation details online.[2]

2: Code:

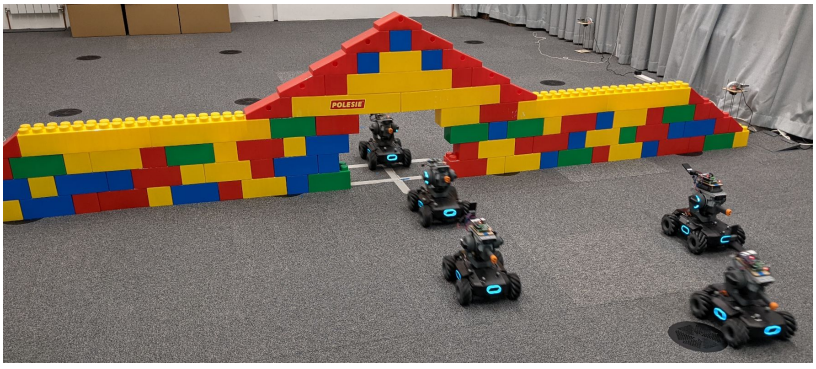


Figure 5.4: We deploy a set of five Cambridge RoboMasters in a real-world setup using GNNs and ad-hoc communication. The robots navigate through a narrow passageway to reconfigure on the other side as quickly as possible.

Environment We define an MRS as graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, 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 $\mathcal{E} = \mathcal{V} \times \mathcal{V}$. At each discrete time step t, each agent $R^i \in \mathcal{V}$ has a position $\mathbf{p}_t^i \in \mathbb{R}^2$, a reference velocity $\mathbf{v}_t^i \in \mathbb{R}^2$, a measured velocity $\hat{\mathbf{v}}_t^i \in \mathbb{R}^2$, and a reference acceleration $\tilde{\mathbf{a}}_t^i \in \mathbb{R}^2$. Let

$$\text{clamp}(a, b, c) = \max(a, \min(b, c)) : \; a \in \mathbb{R}, b \in \mathbb{R}, c \in \mathbb{R}^M, a \leq b$$

where max and min are applied element-wise if $M > 1$. We compute the reference acceleration that is required to apply the reference velocity $\mathbf{v}_t^i$ and constrain it to $a_{\max}$ as

$$\tilde{\mathbf{a}}_t^i = \text{clamp}\left(-a_{\max}, a_{\max}, \frac{\mathbf{v}_t^i - \hat{\mathbf{v}}_{t-1}^i}{\Delta t}\right) \tag{5.1}$$

We constrain the agents from moving into the wall or into other agents (meaning the closest point each agent can approach another agent or the wall is constrained by agents' radius R_{agent}). If the agent would collide by applying the measured velocity, we set it to zero, as follows:

$$\mathbf{v}_t^i = \begin{cases} [0,0]^\intercal & \text{if collision} \\ \text{clamp}\left(0, v_{\max}, \hat{\mathbf{v}}_{t-1}^i + \tilde{\mathbf{a}}_t^i \Delta t\right)) & \text{else,} \end{cases} \tag{5.2}$$

Eventually, the position is updated over Δt as $\mathbf{p}_t^i = \mathbf{p}_{t-1}^i + \hat{\mathbf{v}}_t^i \Delta t$ while constraining it to the environment dimensions $(W_{\text{env}}, H_{\text{env}})$. Note that the reference velocity is dictated by a control policy, and the measured velocity is the velocity with which the agent has moved.

Each agent is assigned a goal position $\mathbf{p}^{i,G} \in \mathbb{R}^2$. Let the set of agents that have reached the goal be $\mathcal{V}_{t=0}^g = \emptyset$. We update this at every time step with the goal reach threshold d_g:

$$\mathcal{V}_t^g = \mathcal{V}_{t-1}^g \cup \{i \mid R^i \in \mathcal{V} \wedge ||\mathbf{p}_t^i - \mathbf{p}^{i,G}|| < d_g\}. \tag{5.3}$$

An episode ends if $R^i \in \mathcal{V}_t^g \; \forall R^i \in \mathcal{V}$ or $t > T_{\max}$, where $T_{\max}$ is the horizon for single episode.

For each episode, the initial positions and the goal positions are initialised randomly. We train with $N = 5$ agents in a plus-shaped formation both for start and goal positions. The relative positions of the agents to each other are always identical, but the starting point, as well as rotation of the initial and goal positions, are randomised according to a uniform distribution chosen so that the initial positions will never be outside the environment boundaries and never in conflict with the wall while covering the whole area on both sides of the wall.

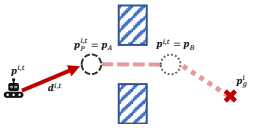


Figure 5.5: To guide each robot through the passage to its goal, we use a shaped reward we refer to as guidance reward. The path is separated into two waypoints to guide the robot towards the goal. As the robot approaches one waypoint, it moves to the next in sequence. The reward is always with respect to the next waypoint.

Reward The objective of each agent is to reach its goal position $\mathbf{p}^{i,G}$ as quickly as possible while avoiding collisions with other agents and the wall. For the agent to reach the goal, we specify two additional waypoints $\mathbf{p}_A$ and $\mathbf{p}_B$ that help each individual agent to be guided to its goal (refer Figure 5.5). Let $\mathbf{p}_t^{i,P}$ be the next waypoint for agent R^i at time t and $\mathbf{d}_t^i = \mathbf{p}_t^i - \mathbf{p}_t^{i,P}$ the vector to the next waypoint. Note that $\mathbf{p}_t^{i,P}$ is initialised to $\mathbf{p}_A$ and will be set to the next waypoint $\mathbf{p}_B$ and eventually, the goal $\mathbf{p}_g^i$ if the current waypoint is approached within a distance $\|\mathbf{d}_t^i\| < d_P$. The

scalar guidance reward $r_t^{d,i}$ is then defined as

$$r_t^{d,i} = \left(\frac{\mathbf{d}_t^i}{\|\mathbf{d}_t^i\|} \cdot \frac{\hat{\mathbf{v}}_t^i}{\|\hat{\mathbf{v}}_t^i\|} \right) \|\hat{\mathbf{v}}_t^i\| \tag{5.4}$$

and intuitively rewards fast movements towards the next waypoints and penalises moving away from it. With this reward, if the agent were to move on some trajectory and reach the point it started on, the cumulative reward would sum to 0. The collision reward $r_t^{c,i}$ is defined as

$$r_t^{c,i} = \begin{cases} -1.5 & \text{if collision with other agent} \\ -0.25 & \text{if collision with wall} \\ 0 & \text{else} \end{cases} \tag{5.5}$$

and the goal-reached reward $r_t^{g,i}$ as

$$r_t^{g,i} = \begin{cases} 10 & \text{if } R^i \in \mathcal{V}_g^t \text{ and } R^i \notin \mathcal{V}_g^{t-1} \\ 0 & \text{else} \end{cases} \tag{5.6}$$

resulting in the total reward $r_t^i = r_t^{d,i} + r_t^{c,i} + r_t^{g,i}$. We found that the highly shaped reward leads to significantly better performance and faster training than a more sparse reward formulation.

Observation and action The observation z_t^i consists of locally available information, specifically the absolute position $\mathbf{p}_t^i$, the relative goal position $\mathbf{p}_g^i - \mathbf{p}_t^i$, as well as a predicted position $\mathbf{p}_t^i + \mathbf{v}_t^i$. Note that the wall is always located at the same absolute position, and our policy would easily generalise to a variable passage position. The reference velocity is the policy's action output $\mathbf{v}_t^i = a_t^i$. We constrain the reference acceleration $\tilde{\mathbf{a}}_t^i$ and velocity $\mathbf{v}_t^i$ to $a_{\max} = 1\text{m/s}^2$ and $v_{\max} = 1.5\text{m/s}$ to aid sim-to-real transfer.

Model If robot R^j is in communication range R_{COM} of robot R^i, it is in robot R^i's neighbourhood $R^j \in \mathcal{N}_t^i$ and robot R^i can emit a message $\mathbf{h}_{0,t}^i$ that is broadcast to its neighbours.

We start with an observation encoder that produces the initial message processed in the GNN as

$$\mathbf{h}_{0,t}^i = \nu^{\text{obs}}(z_t^i).$$

While we used a Convolutional GNN in Chapter 4, we use a message passing GNN [42] in this chapter. This allows us to manipulate the observation more directly and exploit geometric priors from it. Specifically, we use the subtraction operation that helps the neural network to derive

[42]: Gilmer et al. (2017), *Neural Message Passing for Quantum Chemistry*

relative state information from the absolute state provided in z_t^i. The message passing updates the hidden state $\mathbf{h}_{k+1,t}^i$ of each robot R^i for each neural network layer k using the message function ϕ_k^{pre} and the vertex update function ϕ_k^{post} according to

$$\mathbf{h}_{k+1,t}^i = \phi_k^{\text{post}}\left(\sum_{j \in \mathcal{N}_i} \phi_k^{\text{pre}}(\mathbf{h}_{k,t}^i - \mathbf{h}_{k,t}^j)\right). \tag{5.7}$$

Evaluating this message passing GNN is a decentralisable operation depending only on received messages and local information. Note that the subtraction is performed on the embedding $\mathbf{h}_{k,t}^i$ generated from $\nu^{\text{obs}}(z_t^i)$ rather than the raw state z_t^i. While it is not possible to know what information the model is using internally, it is fair to assume that both relative pose and relative velocity are important.

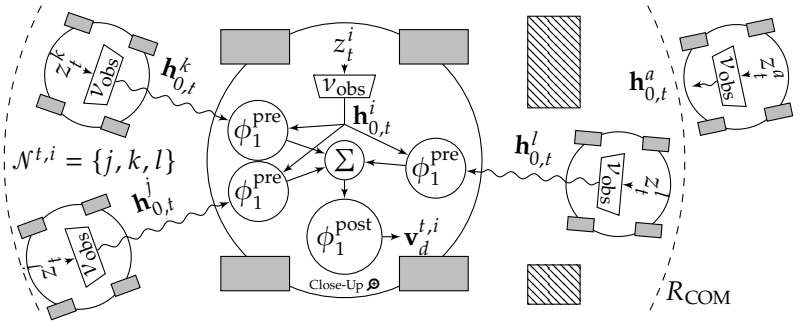


Figure 5.6: The robots form a graph based on their separation and communication range R_{COM}. They leverage communication via latent messages $\mathbf{h}_{0,t}^i$ generated from local observations z_t^i propagated over graph edges (wireless ad-hoc communication links) to overcome the partial observability of the workspace. To solve this task, we utilise and deploy GNN-based policies that aggregate messages of robots within the local neighbourhood $\mathcal{N}_t^i$ and compute a local action.

As the model for the policy π_θ^i, we use the GNN introduced in Section 2.3.2. 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 $K = 1$. The output of the GNN is the embedding $\mathbf{h}_{1,t}^i$, which we convert to a reference velocity with the model ψ^{act} as

$$\mathbf{v}_t^i = a_t^i = \psi^{\text{act}}(\mathbf{h}_{1,t}^i),$$

and the value function from the model ψ^{val} as

$$V_t^i = \psi^{\text{val}}(\mathbf{h}_{1,t}^i).$$

The set of trainable MLPs is $\theta = \{\nu^{\text{obs}}, \phi_1^{\text{pre}}, \phi_1^{\text{post}}, \psi^{\text{act}}, \psi^{\text{val}}\}$. We use the approach described in Chapter 4 to train our model using PPO with local rewards for each agent. We show this model in Figure 5.6.

Experimental setup In total, we run a series of six different real-world experiments for the four modes (Figure 5.2) 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 modelling 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). This Gaussian represents the probability density of message success rate as a function of the distance from the transmitter.

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 customised DJI RoboMaster robots equipped with Raspberry Pi's that locally run policies. The robots are provided with state information as explained in paragraph 5.2.3.

5.4 Results

In this section, we show the results of three experiments. We first evaluate the peer-to-peer communication infrastructure. We then proceed with the ablation study for the case study with decentralised deployment of robots moving through a narrow constriction, and lastly, show more recent results where we deploy eight robots in a centralised mode.

5.4.1 Networking

Multicast and 802.11 retries Figure 5.7 displays unicast-only performance with dashed lines denoting default 802.11 and RTPS settings at 200 messages per second. With default settings, only 44% of all messages are delivered, along with consistently higher delays. Even at messaging rates as low as 20 per second, delays remain highly variable and can exceed the interval between policy executions. We found the reduced latency of multicast operation generally performs better than unicast at similar rates despite lower packet delivery rates. Reducing 802.11 hardware retries to one reduces latency using RTPS unicast defaults (Figure 5.7).

Overall, using multicast, an 802.11 hardware retry setting of one and using RTPS's reliability mechanism results in the most favourable performance, delivering approximately 84% of messages within 20 ms in the Onboard o/Ad-hoc setup, with the remainder being lost. Figure 5.7 highlights the relative latency stability of the chosen scheme where packets are either delivered at low latency or fail to be delivered at all.

Reliability We found disabling positive acknowledgements reduces messaging delays due to the reduction in network contention.

Applicability Any adapter conforming to the same 802.11 revision with similar transmit powers and antennas should perform similarly to the presented results. The achievable inter-robot range is limited by wireless interference. If an increased number of robots contend for airspace, the total number of packets per second achievable will be reduced.

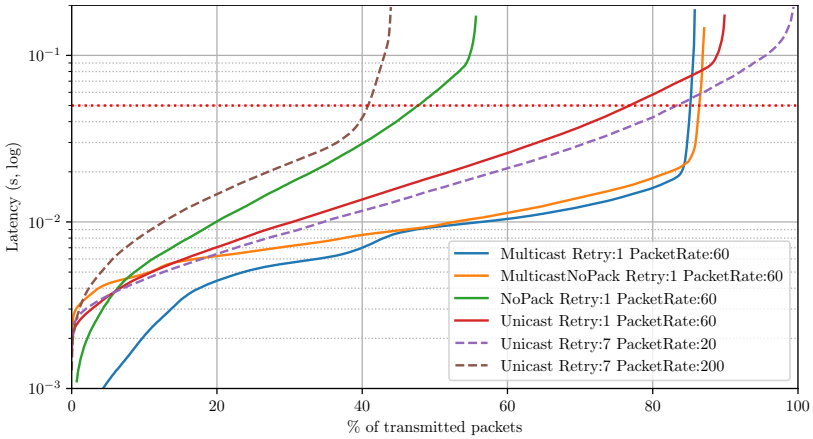


Figure 5.7: CDF of network evaluation results showing the delay between sending and receiving messages via RTPS between ROS nodes. Four scenarios compare Unicast and Multicast performance with and without positive acknowledgements (NoPack), using one 802.11 retry at 60 messages per second (approximate episode load). Additionally, two scenarios show Unicast performance with seven 802.11 retries at 20 and 200 messages per second (default settings). The results are from a 5-agent 802.11 ad-hoc setup, where each agent sends messages to all others, achieving the specified total message rates. The dotted line at 50ms marks the acceptable latency limit.

5.4.2 Narrow constriction

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 Figure 5.8 and show quantitative results in Table 5.1.

Table 5.1: Overview of performance metrics for all case study experiments.

	Success Rate	Median Makespan
Centralised	95.8%	6.3 s
Offboard	90.1%	9.1 s
Onboard o/Infra	85.9%	9.1 s
Onboard o/Ad-hoc	59.4%	9.8 s
Onboard o/Ad-hoc $R_{\text{COM}} = \infty$	70.3%	9.9 s
Onboard o/Ad-hoc Noise	66.7%	10.7 s

The Centralised case reflects the performance gap caused by dynamic constraints that are not considered in the 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 Centralised mode, it features *asynchronous* evaluation but little to no inter-process

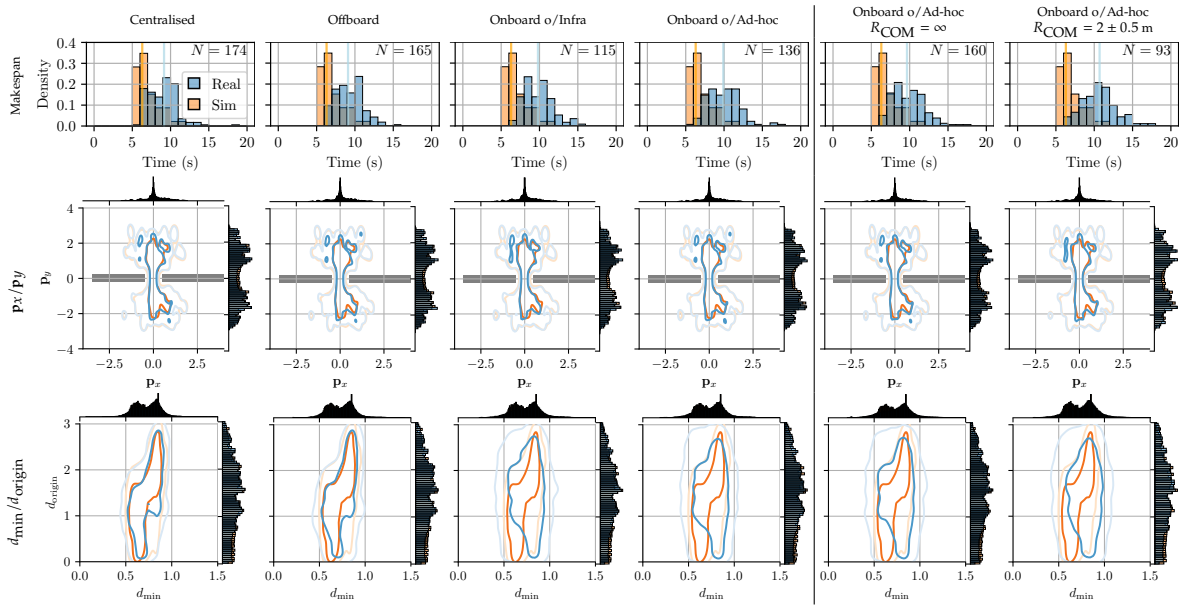


Figure 5.8: We visualise a variety of makespan and position distributions over the six experiments we conducted. The columns show the data of the centralised simulation baseline in orange and the data of the corresponding real-world experiment as labelled in the column headers in blue. For each experiment, we run a total of 192 episodes with 16 different start and goal positions. The last column compares the Onboard o/Ad-hoc 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}.

communication delays, resulting in slightly worse performance of 4.2 pp and worse median makespan of 0.2s wrt the Centralised mode.

The Onboard o/Infra mode moves the decentralised GNN from a central computer to the onboard 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 deterioration of 1.0 s of median makespan w.r.t. the Offboard mode. Onboard o/Ad-hoc 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 neighbourhoods that never occur during training (while the agents are typically fully connected at 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 rows in Figure 5.8 visualise distributions over positions. The second row shows that the distribution of absolute positions over all experiments is consistent, even when comparing to the centralised simulation. In the third row, we compare the distribution of

distance to the origin (or the passage) d_{origin} over the minimum distance between agents d_{min}. While the simulation and real distributions overlap in the Centralised and Offboard modes, 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 decentralised 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.

5.4.3 Scaling to eight robots in VMAS

In the previous experiment, we set the foundations for training multi-agent coordination scenarios. The insights, in particular, related to the simulation of MRS, are the foundations for the Vectorized Multi-Agent Simulator (VMAS) [197] based on Newtonian physics. Simulation is essential in multi-robot research as it enables safety and efficiency in data collection for machine learning pipelines. State-of-the-art simulators employ GPU vectorisation to massively parallelise this process [203], leveraging the Single instruction multiple data (SIMD) paradigm. VMAS is a simulator consisting of a PyTorch differentiable physics engine and a set of challenging multi-robot tasks. Thanks to this, it is possible to rapidly train and deploy multi-robot policies from the simulator to the real world in a zero-shot manner, as demonstrated in previous work [204].

[197]: Bettini et al. (2022), *VMAS: A Vectorized Multi-Agent Simulator for Collective Robot Learning*

[203]: Mittal et al. (2023), *Orbit: A Unified Simulation Framework for Interactive Robot Learning Environments*

[204]: Bettini et al. (2023), *Heterogeneous Multi-Robot Reinforcement Learning*

We provide a model for the RoboMaster platform in VMAS that enables applications beyond prototyping in simulation and deploying in the real world. To generate the reference velocity input $\mathbf{v}_t^i$ required by Freyja, we implement a PID controller layer that regulates the input forces required by VMAS based on these reference velocities. This layer allows us to train neural network policies that output reference velocities $\mathbf{v}_t^i$ and directly use them on our physical platform without any change. Secondly, a key to the real-world deployment was an iterative fine-tuning process for some of the physics parameters in VMAS. These are parameters that depend on the robot platform and its interaction with the environment, such as friction coefficients, drag, control ranges, state boundaries, and robot geometry. This tuning process can be done once for each robot-environment pair and can be then utilised for any subsequent deployments.

We demonstrate the zero-shot deployment of MARL policies for a multi-robot navigation task trained in VMAS using the BenchMARL library [205] and TorchRL [206]. Specifically, we train a multi-robot navigation policy that is essentially similar to the experiment outlined in Section 5.3.2. There are no static obstacles (like the wall with a passage outlined in the previous experiment). Agents are initialised at random positions and have a random goal position they have to navigate to. The model and training procedure are identical. We increase the velocity limit to $2m/s$ and the acceleration limit to $2m/s^2$, allowing slightly more agile actions.

[205]: Bettini et al. (2023), *BenchMARL: Benchmarking Multi-Agent Reinforcement Learning*

[206]: Bou et al. (2024), *TorchRL: A data-driven decision-making library for PyTorch*

After training the policy, we provide qualitative experiments for four in Figure 5.9 and eight robots in Figure 5.10, both in the centralised deployment mode.

Supplementary material

We provide the following supplementary videos:

Square: Line: Eight:

In the four-robot experiment reported in Figure 5.9, we initialise the robots to start at the corners of a 2 m×2 m square and assign goal positions diagonally to each robot. The robots coordinate and solve this problem by moving in a circle. Note that this is completely emergent behaviour – the robots were trained to handle random start and goal positions, and this specific configuration is one possible random perturbation of this. The policy learned that the optimal behaviour for the whole team is to move in a roundabout.

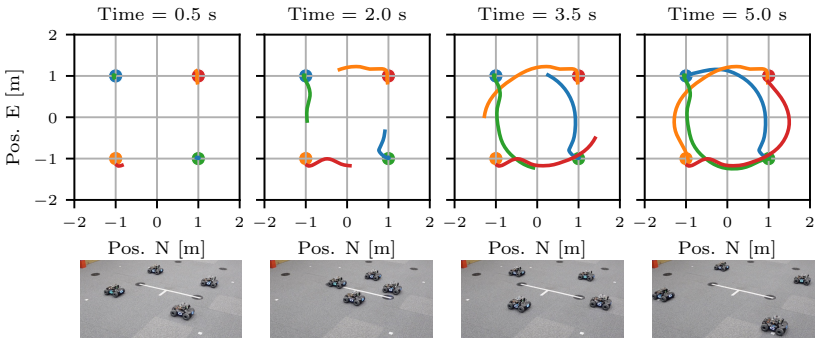


Figure 5.9: Four agents (colour-coded) start at the corners of a square and attempt to swap positions with the diagonally opposite agent, which leads to a conflict at the origin. The lines indicate their trajectories up to a particular time point, and the circles indicate their corresponding goals. Agents are able to seamlessly execute the learned policy in the previously unseen real-world environment. We show snapshots from the real-world setup at the bottom.

In the eight-robot experiment reported in Figure 5.10, we initialise the robots to start on two opposing sides of a 4 m × 2 m rectangle. The two groups of four inner and four outer robots have to swap positions with their two swapped opposing robots. We show that the policy can handle this, too. This is the first experiment of its kind to show the real-world transfer of GNN-based control policies on up to eight robots.

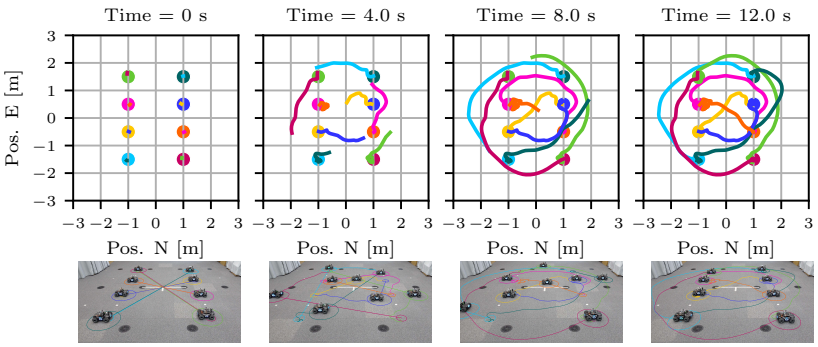


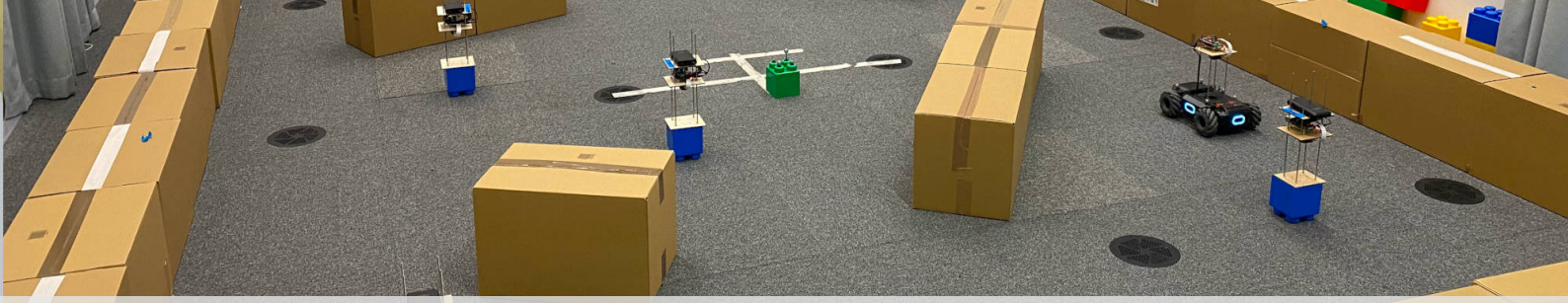
Figure 5.10: Eight agents (colour-coded) start on opposite sides of a 4 m × 2 m rectangle and attempt to swap positions with agents on the respective other side, which leads to multiple conflicts. The lines indicate their trajectories up to a particular time point, and the circles indicate their corresponding goals.

5.5 Conclusion

We are the first to demonstrate the real-world deployment of a GNN-based policy to a decentralised real-world MRS using ROS2 and an ad-hoc communication network. We performed a suite of experiments that discussed the selection of suitable networking settings and subsequently presented results on a real-world scenario requiring tight coordination amongst robots. Our results showed that our framework allows for the successful deployment of our control policy in an ad-hoc 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 centralised 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. In the future, we plan to 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 onboard sensing.

Learning communication for cooperative perception



6 Cooperative perception for visual navigation

In the first part of this thesis, we focussed entirely on *controlling* MRS. We assumed that ground-truth state information is available, but this assumption is false for many real-world systems. Robots must rely on local sensing (which can be combined with communication) for a truly decentralised deployment. Therefore, in the second part of this thesis, we focus on how simple monocular cameras can be integrated into this framework. Specifically, in this chapter, we consider the problem of using a *positioning-free* visual sensor network that *learns* to guide a mobile robot to some target.

Efficiently finding and navigating to a target in complex unknown environments is a fundamental robotics problem, with applications to search and rescue [207], surveillance [208], and monitoring [209]. We focus on navigating in an unknown environment to find a target under the guidance of a *visual sensor network*. Prior work has provided effective solutions employing low-cost wireless sensors to guide robotic navigation [209, 210]. These studies demonstrate that at a small additional cost—i.e., the deployment of cheap sensors with local communication capabilities—the requirements for the robot's capabilities can be significantly reduced while simultaneously improving its navigation efficiency.

However, the implementation of traditional sensor network guided navigation is cumbersome, as it commonly requires that *(i)* robots and sensors have access to external positioning systems (e.g., GPS) and that *(ii)* an explicit map is built (that the robot can then use to plan a path). This approach's main drawback is that sensors and robots must always accurately estimate their positions within a global frame of reference (e.g., through an available global positioning system such as GPS). It also requires designing a multi-hop communication strategy that allows the sensors to jointly build and maintain an up-to-date map of the environment.

Successful navigation requires the robot to aggregate and process information received from nearby sensors, which helps it decide which direction to move in. Towards this end, we propose a data-driven approach in which the nodes in the network learn a policy that incorporates visual processing and a communication module. We propose a GNN-based architecture that learns to encode and communicate relevant viewpoint

6.1 Contributions 82
6.2 Approach 82
6.2.1 Visual navigation using sensor networks 83
6.2.2 Zero-shot real-world transfer 88
6.3 Results 90
6.3.1 Performance metrics . . . 91
6.3.2 Simulations 91
6.3.3 Sim-to-real policy transfer 93
6.4 Conclusions 95

[207]: Mirowski et al. (2017), *Learning to Navigate in Complex Environments*

[208]: Kolling et al. (2009), *Surveillance Strategies for Target Detection with Sweep Lines*

[209]: Corke et al. (2005), *Networked Robots: Flying Robot Navigation using a Sensor Net*

[209]: Corke et al. (2005), *Networked Robots: Flying Robot Navigation using a Sensor Net*

[210]: Kashino et al. (2020), *A Hybrid Strategy for Target Search Using Static and Mobile Sensors*

information to the mobile robot. During navigation, the robot is guided by a model that we train through IL to predict the effective cost-to-go (to the target), thereby approximating optimal motion primitives. The setup is depicted in Figure 6.1. Our experiments demonstrate successful sim-to-real transfer to a laboratory setup. We also showcase the ability of our network to adapt to dynamic changes in its configuration by showing examples of successful navigation to the target while sensor nodes are being moved.

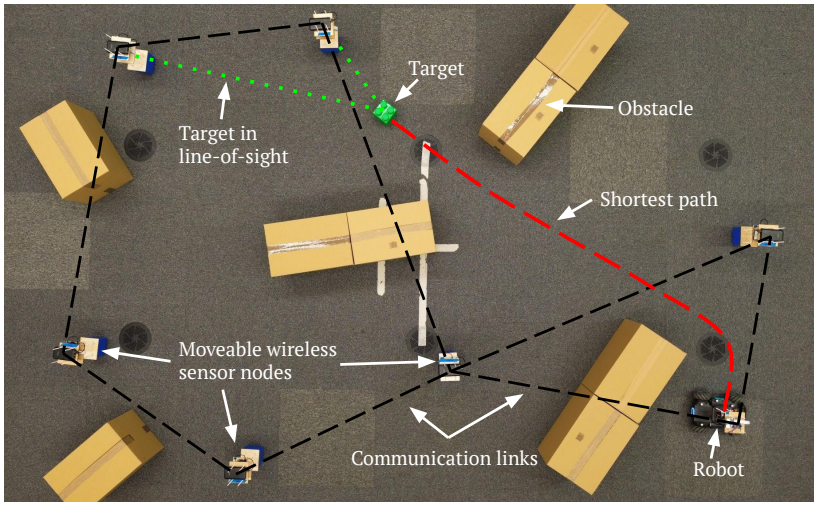


Figure 6.1: Our setup consists of a workspace populated with a target, obstacles, a mobile robot, and movable wireless sensor nodes equipped with cameras. We train a GNN to guide the robots toward the target along the shortest path. Our setup is positioning-free, relies on visual information and communication only, and therefore allows for dynamic online reconfiguration of all components.

6.1 Contributions

Supplementary material

We provide the following supplementary material for the chapter:

Code: Video:

Our contributions include the following:

- 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 or relative positioning information*.
- We provide an end-to-end visual navigation policy that leverages GNNs to *learn what needs to be communicated* (among sensors and robots) 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).

6.2 Approach

Our setup consists of a cluttered environment, a mobile robot, and a set of visual sensors. The continuous environment $\mathcal{W}$ contains a set

of randomly placed obstacles $\mathcal{C} \subset \mathcal{W}$ and N randomly placed visual sensors $\mathcal{S} = \{S^1, \ldots, S^N\}$. At every time step t, each sensor S^i is capable of taking an omnidirectional RGB image z_t^i of its surrounding environment (equirectangular projection of a fisheye lens) but has *no positioning information*. Each sensor S^i can communicate with nearby sensors within communication range, i.e., $S^j \in \mathcal{N}_t^i$. A target object G is located randomly in the 2D ground plane at position q^G. Each sensor predicts its direction $u_t^i \in \mathcal{U}$ along its shortest path towards the target G, where $\mathcal{U}$ is a set of H possible discrete directions. The mobile robot R is located at position q_t^R and moves in the ground plane in $\mathcal{W} \setminus \mathcal{C}$. 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 $a_t \in \mathcal{A}$ 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 utilises 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 formalise this as a sequential decision-making problem under uncertainty about the underlying world [211] and define a corresponding MDP. At time step t, let $s_t \in \mathcal{O}$ be the observed state of the environment, i.e $s_t = \{z_t^1, \ldots, z_t^N\}$. On executing a_t, the new state s_{t+1} is determined by the underlying world $\mathcal{W}$, which is a hidden variable sampled from a prior $P(\mathcal{W})$ 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.

[211]: Bhardwaj et al. (2017), *Learning Heuristic Search via Imitation*

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 time horizon T is reached. Given a prior distribution over worlds $P(\mathcal{W})$ 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

$$V(s_t) = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_\pi^t} [c(s_t, \pi(s_t))] \tag{6.1}$$

where $d_\pi^t = P(s_t|\pi, \mathcal{W}, q_t, q^G)$ is the distribution over states induced by running π on the problem $(\mathcal{W}, q_t, q^G)$ for T steps [212], and we evaluate the performance of a policy as

[212]: Ross et al. (2014), *Reinforcement and Imitation Learning via Interactive No-Regret Learning*

$$J(\pi) = \mathbb{E}_{\substack{\mathcal{W} \sim P(\mathcal{W}), \\ (q_0^R, q^G) \sim P(q_0^R, q^G)}} [V(s_t)]. \tag{6.2}$$

6.2.1 Visual navigation using sensor networks

This section describes our approach to training the navigation policy π. Figure 6.2 shows the simulation environment and Figure 6.4 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 z_t^i and the messages shared by other sensors. We use a variant of IL using expert solvers (i.e., cheap and fast path planning algorithms).

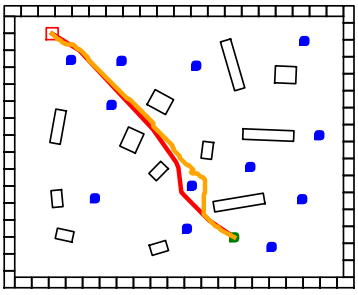


Figure 6.2: Our setup consists of an environment populated with obstacles, visual sensors (blue), a mobile robot equipped with a sensor, and a green target, where the mobile robot has to find the shortest path (red, taken path orange) to an occluded target by incorporating communicated sensor information to enhance its navigation performance.

Our method does not rely on any positioning information whatsoever, and the neighbourhood $\mathcal{N}_t^i$ 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 neighbour set is defined as $\mathcal{N}_t^i = \{S^j | L(S^i, S^j) \leq D_S\}$, where $L_t(S^i, S^j)$ is the euclidean distance between S^i and S^j at time t, and D_S is the communication range.

We explain how we generate the dataset

$$\mathcal{D}_{\text{IL}} = \{(o_0^{i,m}, \hat{A}_0^{i,m})\}_{m=1,i=1}^{M,N}$$

consisting of M per-sensor cost-to-go advantage labels $\hat{A}$ and observations o. We also detail the neural network model consisting of the feature extractor ν, the feature aggregator ϕ and the post-processor ψ to predict the advantages. The architecture is depicted in Figure 6.4. This neural network is optimised with the objective J_{IL} as

$$J_{\text{IL}}(\psi \circ \phi \circ \nu) = \mathbb{E}_{o,\hat{A}\sim\mathcal{D}_{\text{IL}}} \left[\|\psi \circ \phi \circ \nu(z) - \hat{A}\|^2 \right]. \tag{6.3}$$

While we train our policy in simulation, we take an approach that facilitates sim-to-real transfer (see Section 6.2.2).

Data generation

IL can be treated as a supervised learning problem. Since future states are influenced by previous actions in IL, the i.i.d. assumption of supervised learning is invalidated. AggreVate [213] 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 a standard supervised learning problem with an i.i.d. assumption for each sample.

[213]: Ross et al. (2010), *Efficient Reductions for Imitation Learning*

Let Q^i be the cost-to-goal for sensor S^i. Let the set of possible directions be

$$\mathcal{U} = \{u^1, \ldots, u^H\}.$$

Let $Q_{u,k}^i$ be the cost-to-goal for sensor S^i upon moving towards direction u_k^i. For each environment, we construct a visibility graph, in which the edges of obstacles, sensors and robots are nodes that are connected by an

edge if there is no line-of-sight interaction between any two nodes. We then use Dijkstra's shortest path algorithm on these visibility graphs to compute cost-to-goal values Q^i. We visualize this in Figure 6.3.

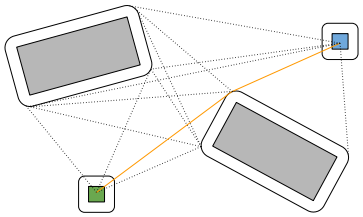


Figure 6.3: We construct a visibility graph to compute the shortest path from a blue sensor to a green target considering grey obstacles with a collision boundary.

To simplify the training procedure and facilitate better generalisation, instead of learning absolute cost-to-goal values Q^i, we learn a relative cost-advantage vector

$$\mathbf{A}^i = \begin{bmatrix} Q^i_{a,1} - Q^i \\ \vdots \\ Q^i_{a,k} - Q^i \end{bmatrix}$$

that subtracts the cost from the current state to the goal for each of the H possible directions and thus is scale-invariant (since the scale of the cost-to-go values is not relevant for a sequential decision-making problem).

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 [214] computing a path $\mathcal{P}(q^R_0, q^G)$. If no valid path can be found for any of the sensors or robots, the map is discarded.

[214]: Coleman (2012), *Lee's O (n2 log n) Visibility Graph Algorithm Implementation and Analysis*

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 a size of 0.5 m × 0.3 m × 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 generate the dataset $\mathcal{D}_{\text{IL}}$ consisting of $M = 40,000$ environment samples for $N = 7$ sensors.

GNN-based feature aggregation across sensor network

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

Local feature extraction. We first use a Convolutional Neural Network (CNN) $\nu(\cdot)$, specifically MobileNet v2 [215], which has been pre-trained on ImageNet, to extract features $\mathbf{h}^i_{0,t}$ from the image z^i_t of each sensor S^i. MobileNet is optimised 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 Section 6.2.2 and visualised in Figure 6.4. The encoding $\mathbf{h}^i_{0,t}$ has a size of $\mathbb{R}^{32}$ and is communicated to other sensors within the communication range.

[215]: Sandler et al. (2018), *MobileNetV2: Inverted Residuals and Linear Bottlenecks*

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 GNNs-based feature aggregation for information prediction [216] or robot coordination tasks [3, 94]. Specifically, in the aforementioned papers, each agent only needs to aggregate information from the nearest few neighbours 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 outside line-of-sight). 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 neighbours 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.

[216]: Akter et al. (2019), *Improving IoT Predictions through the Identification of Graphical Features*
[3]: Li et al. (2020), *Graph neural networks for decentralized multi-robot path planning*
[94]: Hu et al. (2021), *VGAI: End-to-end learning of vision-based decentralized controllers for robot swarms*

The GNNs model $\phi^{(L)}(\cdot)$ consists of K layers and generates an encoding $\mathbf{h}^i_{k,t}$ for each layer K. It takes the image encoding generated by the encoder $\mathbf{h}^i_{0,t}$ as input to the first layer, aggregates communicated neighbours' features using the neighbourhood $\mathcal{N}^i_t$ over multiple layers, and extracts fused features $\mathbf{h}^{i,K}_{0,t}$ where $\mathbf{h}^{i,L}_{0,t}$ is the encoding generated by the last GNNs layer for each sensor S^i so that $\phi^{(L)}(\mathbf{h}^i_{0,t}) = \mathbf{h}^i_{K,t}$.

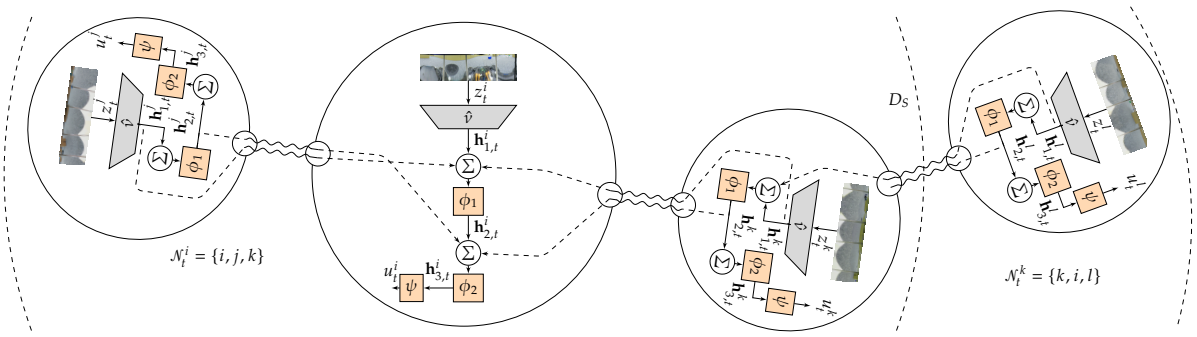


Figure 6.4: Overview of our model, with communication visualised for agent R^i. We train a decentralised GNNs-based policy that takes camera observations z_t^i as input and outputs directions towards the target u_t^i for each node. The CNN $\hat{\nu}$ encodes images into features $\mathbf{h}_{0,t}^i$ and the GNNs ϕ further encodes this for multiple communication hops in K layers as $\mathbf{h}_{K,t}^i$ (here, $K = 2$). Lastly, the post-processing MLP $\psi(\cdot)$ generates cost-to-go advantages $\mathbf{A}_t^i$ 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. Modules trained in simulation and the real world are coloured orange and grey, respectively. Note that all sensor nodes are trained to predict the direction to increase the number of samples observed, but during deployment, only the robot uses this information.

A GNNs model consists of a stack of neural network layers, where each layer aggregates local neighbourhood information, i.e., features of neighbours, around each node and then passes this aggregated information on to the next layer. Specifically, our method builds on the GNNs layer introduced in [217]. We use a multi-layer message passing GNNs where each layer $K > 0$ computes new features as

[217]: Morris et al. (2019), *Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks*

$$\phi_k(\mathbf{h}_{k,t}^i) = \mathbf{h}_{k+1,t}^i = \phi_k^{\text{post}}\left(\mathbf{h}_{k,t}^i, \sum_{j \in \mathcal{N}_i} \phi_k^{\text{pre}}(\mathbf{h}_{k,t}^j)\right). \tag{6.4}$$

that are communicated in the neighborhood $\mathcal{N}_t^i$. The functions ϕ_k^{pre} and ϕ_k^{post} are trainable MLPs.

Cost-to-goal prediction. Lastly, we utilise a post-processing Multi-Layer Perceptron (MLP) to predict a set of cost-to-go advantages for each sensor so that $\psi(\mathbf{h}_{K,t}^i) = \mathbf{A}_t^i$, which is eventually used to sample a target direction u_t^i and eventually transform it to an action a_t.

Note that this is the action for the one mobile robot, and therefore not indexed as one of many agents i.

Policy. This results in the composition $\psi \circ \phi \circ \nu(z_t^i) = \mathbf{A}_t^i$ to compute a cost-to-go advantages from the local image z_t^i and features $\mathbf{h}_{0,t}^{i,k}$ communicated through the neighborhood $\mathcal{N}_t^i$. We model the set of target directions $\mathcal{U}$ as a discrete distribution so that $u_t^i = \text{cat}(\sigma(-\alpha \mathbf{A}_t^i))$ where $\text{cat}(\cdot)$ samples from a categorical distribution, $\sigma(\cdot)$ is the softmax function, and α is a hyperparameter to adjust the stochasticity, which may help the robot to avoid oscillating between two possible actions. 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 centralised MDP relying on a central state $s_t = \{z_t^1, \ldots, z_t^N\}$

consisting of the observation of all sensors. Therefore, we can define the policy π as

$$\begin{aligned}\pi(\{z_t^1, \ldots, z_t^N\}) &= \text{cat}(\sigma(-\alpha(\psi \circ \phi \circ \nu(z_t^1))))\beta \\ &= u_t^1 \beta \\ &= a_t,\end{aligned}$$

where β is a hyperparameter to scale the magnitude and thus transform a direction into a velocity action $a_t \in \mathcal{A}$. Note that even though the notation of the policy is centralised, the execution is inherently decentralised, and instead of depending on raw image observations for other sensor nodes, the robot only depends on the local observation z_t^1 and latent encodings communicated through the GNNs ϕ.

Training. We use supervised learning to predict the pre-computed cost-to-go advantages. We predict $H = 8$ advantages (unless specified otherwise). The minibatch size is 16. The dataset contains 40000 samples, of which we use 80% for training, 29% for evaluation and 1% for testing. In this chapter, the test set is used for all evaluations and path visualisation. 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 are 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.

6.2.2 Zero-shot real-world transfer

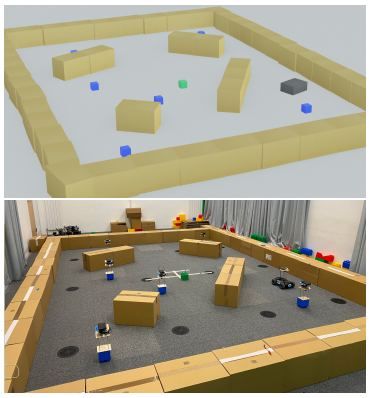


Figure 6.5: Our framework leverages a setup consisting of a simulation environment that corresponds to the real-world setup.

To demonstrate the feasibility of a zero-shot transfer of our model, we design a twin environment setup consisting of both real and digital copies of the operational space, see Figure 6.5. The real setup includes custom-built sensors that provide local visual sensing to nearby nodes within the communication network. Later, in Section 6.3, we report results that demonstrate the effectiveness of our sim-to-real transfer approach for the real-world scenario.

Setup

Twin environments The real-world environment is 5.7 m × 4.2 m, and is cluttered with three to five obstacles of different sizes. We use standard cardboard boxes for obstacles and blue and green building blocks to identify sensor locations and target locations, 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 [218] and is illustrated in Figure 6.5.

[218]: Michel (2004), *Webots: Professional Mobile Robot Simulation*

Custom-made sensor nodes We design and construct six sensor nodes that consist of a contraption holding the downward-facing camera with a fisheye lens as well as a local data processing unit.

Mobile robot We use the DJI RoboMaster as a mobile robot platform. A seventh sensor node that also serves as the controller for the robot is mounted on top. We visualise this setup in Figure 6.6. The robot employs a collision shielding mechanism, that takes as input distance measurements to detect the near-sided obstacles and the border of the environment. Repulsive force against detected obstacles is generated through a potential field [219]. In practice, during real-world deployment, we have a fully connected communication network due to wireless communication ranges exceeding the size of our indoor environment.

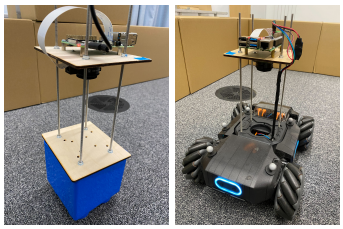
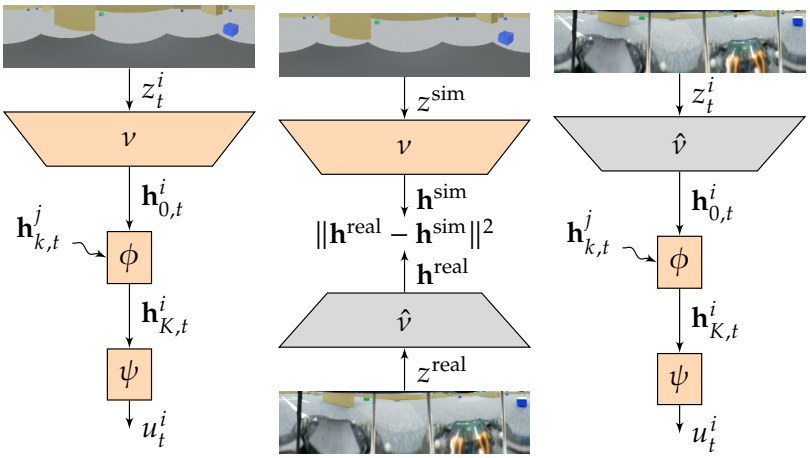


Figure 6.6: The workspace is endowed with custom-built sensors with fisheye cameras that are capable of communicating with each other. One sensor is attached to a mobile robot that acts on the output of that sensor.

[219]: Hwang et al. (1992), *A Potential Field Approach to Path Planning*

Domain adaptation

As outlined in Figure 6.4, we first train the policy using simulated images through IL. Figure 6.7 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 teleoperate 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 $\mathcal{D}_{\text{s2r}} = \{(o^j_{\text{real}}, o^j_{\text{sim}})\}^M_{m=0}$.



(a) Sim deployment **(b)** Translator training **(c)** Real deployment

Figure 6.7: Our sim-to-real transfer framework. (a) We first train and deploy the policy in simulation. (b) We proceed to collect image pairs from simulation z^{sim} and the real world z^{real} and use them for training a real-to-sim translator model $\hat{\nu}$ (grey) by reconstructing latent features from the simulation domain z generated through the encoder trained in simulation ν (orange) with real-world images by minimising the loss function J_{s2r}. (c) We combine the translator model $\hat{\nu}$ trained on real-world images (grey) with ϕ and ψ trained in simulation (orange) to deploy the policy to a real-world setup.

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 × 0.3 m × 0.3 m as obstacles and coloured building boxes of size 0.1 m × 0.1

m × 0.1 m as sensor and target indicator. As a 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, projection from the raw image to equirectangular [220] images, compression and streaming at a frame rate of 12 Hz. We provide additional details on the camera calibration in Section A.3.2.

[220]: Zhang (2000), *A Flexible New Technique for Camera Calibration*

In total, we constructed eight different environments, each populated with six 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 so that the sensor images are only 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.

[170]: Zhang et al. (2019), *VR-Goggles for Robots: Real-to-Sim Domain Adaptation for Visual Control*

We use a similar approach to [170], 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 $\hat{\nu}(\cdot)$ that maps real images z^{real} to the encoding of the corresponding simulated image $\nu(o^{\text{sim}})$ by minimizing the objective $J_{\text{s2r}}(\cdot)$ while keeping $\nu(\cdot)$ fixed,

$$J_{\text{s2r}}(\hat{\nu}) = \mathbb{E}_{z^{\text{sim}}, z^{\text{real}} \sim \mathcal{D}_{\text{s2r}}} \left[\|\hat{\nu}(z^{\text{real}}) - \nu(z^{\text{sim}})\|^2 \right]. \tag{6.5}$$

Training the translator is done in settings similar to 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 colour 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 shear of ±3° to the real image.

6.3 Results

We first introduce the metrics we use for evaluation and then demonstrate the performance of our method in simulation based on the methodology

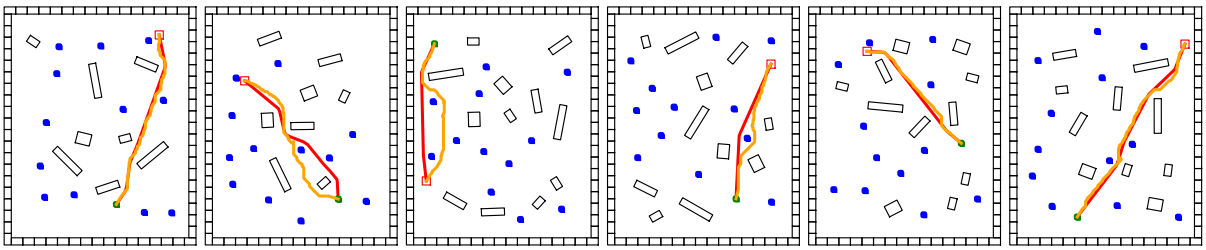


Figure 6.8: A selection of policy evaluations for a policy trained on a small environment ($W = 8$ m, $H = 10$ m, $N = 7$) and evaluated on a large environment ($W = 16$ m, $H = 20$ m, $N = 13$). Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's 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. The policy generalises.

introduced in Section 6.2.1. Second, we demonstrate successful zero-shot transfer to the real-world setup introduced in Section 6.2.2.

6.3.1 Performance metrics

We evaluate the trained policies on the unseen test split of the training dataset as well as a generalisation 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 *success rate* is the fraction of all successful runs as

$$\frac{1}{M}\sum_{m=1}^{M} C_m.$$

We furthermore report Success weighted by path length (SPL) [221], 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 is the length of the path actually taken. We report all results for environment configurations where the target is outside line-of-sight, therefore requiring additional sensor coverage for an efficient solution.

[221]: Anderson et al. (2018), *On evaluation of embodied navigation agents*

6.3.2 Simulations

We train five variants of our policy to evaluate our approach. We first train a policy with communication range $D_S = 0.0$ m as a baseline (i.e., no communication). We also train three policies with communication ranges of $D_S = 2.0$ m, $D_S = 4.0$ m and $D_S = \infty$ m (fully connected) respectively, all with a single GNNs layer. Lastly, we train a policy with a communication range of $D_S = 2.0$ m and two GNNs layers. The results can be seen in Table 6.1.

		Training Distr.		Generalization	
D_S	K	Success	SPL	Success	SPL
0.0	1	0.827	0.719	0.619	0.492
2.0	1	1.000	0.925	0.833	0.742
4.0	1	0.981	0.912	0.905	0.804
2.0	2	0.962	0.885	0.857	0.756
∞	1	0.962	0.880	0.952	0.853

Table 6.1: Evaluation of sim result for policies trained for different communication ranges D_S and number of GNNs layers K.

We evaluate all policies on unseen maps on the small test set (environment sized $W = 8$ m and $H = 10$ m with $N = 7$ sensors, $M = 52$ trials) and the large generalisation set (environment sized $W = 16$ m and $H = 20$ m with $N = 13$ sensors, $M = 42$ trials). The baseline policy without communication has a success rate of 0.827 and an SPL of 0.719 in the small environment. These values are the lower performance bound for no 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 $D_S = 2$ m and $K = 1$. The policies with $D_S = 2.0$ m and $K = 2$ perform similarly to the policy with $D_S = 4$ m. The policy with $D_S = \infty$ m and $K = 1$ performs slightly worse. This can be attributed to the unfiltered inclusion of information from all sensors. Adding locality through a neighbourhood and considering multiple neighbourhoods through multi-hop communication helps in building a more appropriate global representation.

We 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. Samples from this evaluation are visualised in Figure 6.8.

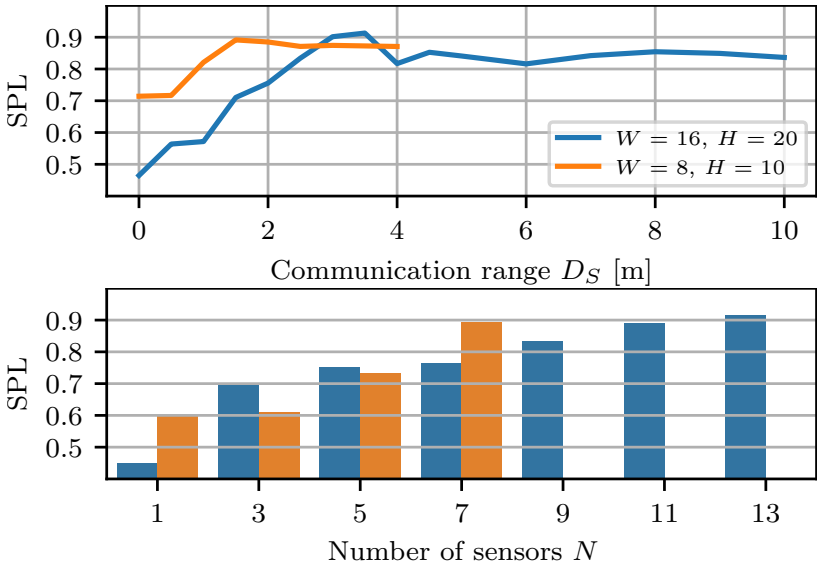
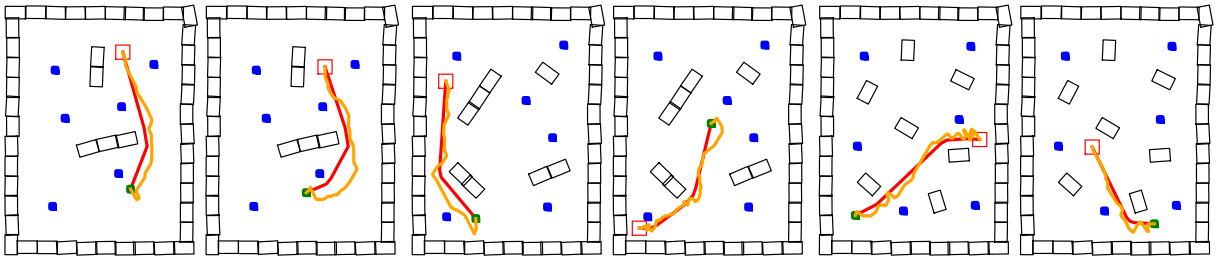


Figure 6.9: Top: We record the performance of the policy trained for $D_S = 2$ m and $K = 2$ for a range of different communication ranges and number of sensors. The large environment (blue) is populated with $N = 13$ sensors, and the small environment (orange) with $N = 7$ sensors. Bottom: The large environment (blue) has a communication range of $D_S = 3.5$ m and the small environment (orange) of $D_S = 1.5$ m. Both values result in peak performance for the maximum number of sensors in each environment. Increasing the communication range and number of sensors benefits the SPL.

In Figure 6.9, we further analyse the benefit of communication on our method. We use the policy trained for $D_S = 2$ m and $K = 2$ 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 Table 6.1 for the large environment are suboptimal due to the constrained communication range, since $D_S = 2.0$ m for $K = 2$ 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 $D_S = 0.0$ m to 0.91 for $D_S = 3.5$ m and then slightly



(a) Env. A **(b)** Env. B **(c)** Env. C

Figure 6.10: A selection of policy evaluations for each real-world environment. Blue squares indicate sensor positions, green squares the target position q^G, red squares the robot's initial position q_0^R, red paths the shortest path computed by the expert P and the orange path the path p chosen by the policy π.

decreases to a constant of 0.85 for $D_S \geq 4.0\,\text{m}$, resulting in a performance increase of 2.0× 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× 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).

Lastly, we compare the effect a varying number of cost-to-go advantages have on the policy. We compute $H = 8$ and $H = 16$ cost-to-go advantages and train two policies. The results can be seen in Table 6.2 and show that the overall success rate increases and detour decreases with a larger H.

	LOS		NLOS	
H	Success	SPL	Success	SPL
8	1.000	0.945	0.837	0.711
16	1.000	0.967	0.913	0.801

Table 6.2: Analysis of the effect of the amount of cost-to-go advantages on performance.

6.3.3 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 Section 6.2.2. We randomly construct three different environments and run between 19 and 29 runs per environment, resulting in a total of 69 evaluation runs. We visualise a selection of paths in Figure 6.10.

We first perform experiments to showcase the ability of the policy to dynamically respond to changes in sensor positions, which is useful in cases where the sensors are mounted on mobile platforms Figure 6.11. The trials show that the robot is not able to locate the target if it is not within line-of-sight of at least one sensor (see the (a), (e) and (i)).

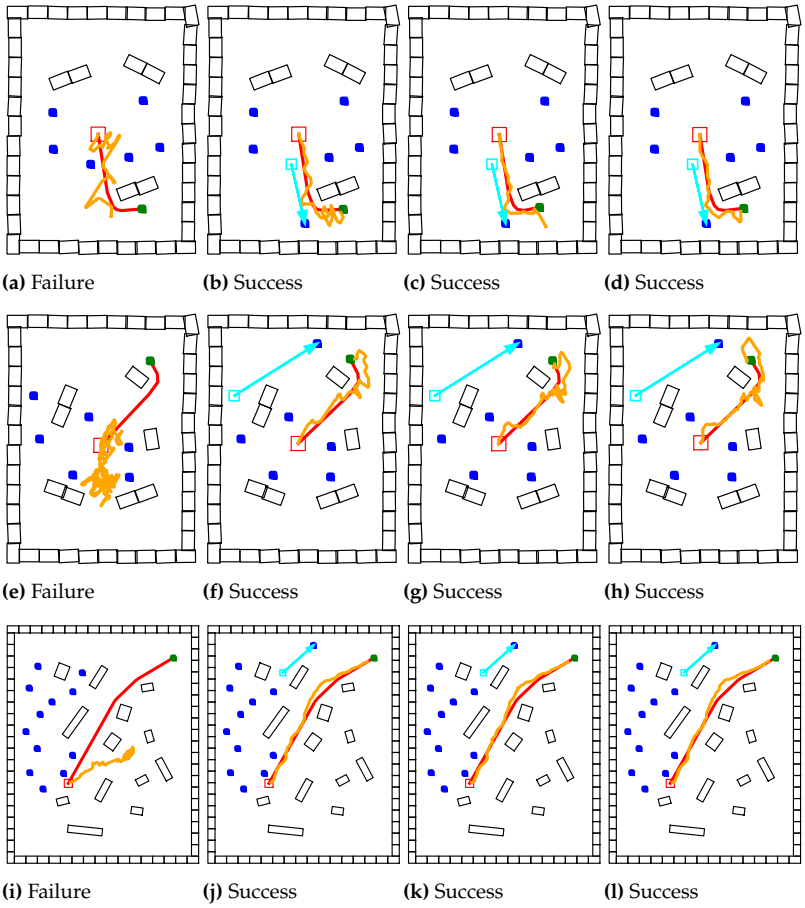


Figure 6.11: We showcase the capability of our method to adapt to changing sensor layouts. We first construct a real-world environment in which no sensor has visual coverage of the target (a), (e), and (i). The robot is not able to locate the target. We then move a sensor within line-of-sight to the target (cyan). The robot is now able to infer its direction w.r.t. the target and is, therefore, able to navigate towards it ((b-d), (f-h) and (j-l)). The first two rows are performed in the real-world setup, and the last row in simulation.

As soon as one sensor is moved within line-of-sight to the target, the robot is able to successfully navigate to the target. Our experiments furthermore indicate that our approach is able to handle dynamically changing obstacle positions, as long as they are observed by a sensor.

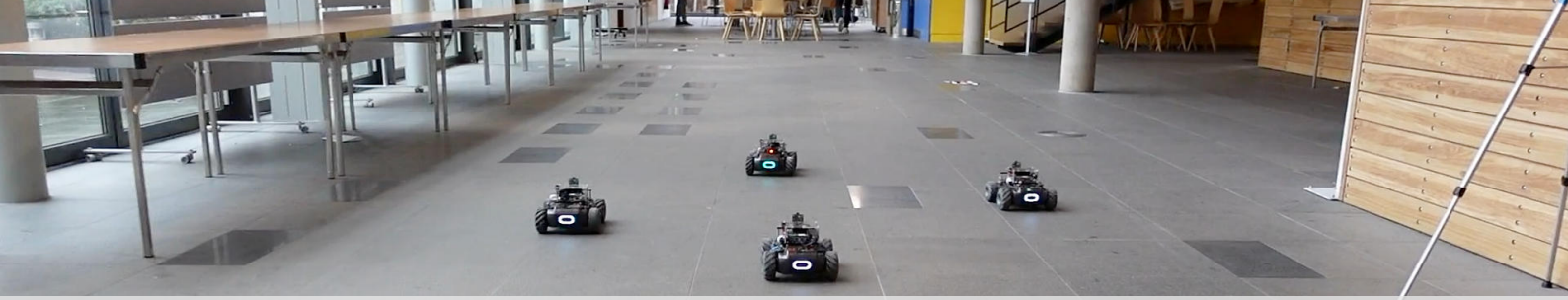
Env	M	Success	SPL
A	29	1.000	0.588
B	19	0.947	0.634
C	21	0.905	0.502
All	69	0.957	0.574

Table 6.3: Evaluation of real-world results for three different environments for the fully connected policy. We report the number of trials M for each environment.

The results for all environments are shown in Table 6.3. The total success rate across all environments is 0.957 and 0.574 SPL. Environment A is the least challenging environment, with only two obstacles in total, and has the highest success rate of 1.0 and SPL of 0.588. Environment C is, with a success rate of 0.905, the most challenging one, with a total of seven obstacles, ranging from small to large, and narrow passages between obstacles. Environment B has more obstacles than Environment B but fewer than Environment C, with a success rate of 0.947 and an SPL of 0.634. Even though the sim-to-real transfer is generally successful, there is a noticeable reality gap. This can be attributed to the physics reality gap, which has not been fully addressed in this chapter, which causes oscillations when moving in the real world. Even though we did not conduct any real-world baseline experiments, it is to be expected that the SPL and success rate for $D_S = 0$ (i.e., no communication) is significantly lower, as observed in our simulation results.

6.4 Conclusions

We propose a vision-only-based learning approach that leverages a GNN to encode and communicate relevant viewpoint information to the mobile robot. In our experiments, we first demonstrate generalisation 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× improvement in small environments and a 2.0× 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, or pre-calibration of the sensor network, which we demonstrate in real-world experiments. The benefit of utilising 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 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.



7 Visual spatial foundation models

In the first part of this thesis, we focussed entirely on *controlling* MRS. In the previous chapter, we lifted that constraint by considering training a *positioning-free* visual sensor network to guide a mobile robot to some target, using only visual information in a controlled laboratory environment. In this chapter, we go one step further and propose a perception model foundational for a wide range of multi-robot control tasks.

7.1 Contributions 98
7.2 Approach 98
7.2.1 Model 99
7.2.2 Training 102
7.3 Results 105
7.3.1 Simulation 105
7.3.2 Real-world 108
7.3.3 Real-world pose control 112
7.4 Conclusion 115

Spatial understanding is a cornerstone of robotic operation in unstructured environments, relying on effective pose estimation and environmental perception [222–229]. While GNSS, LiDAR, and UWB sensors have been instrumental in advancing robotics, they are limited by constraints such as indoor operation and unreliability when operating around reflective surfaces or in bright environments. *In contrast, color cameras offer a low-cost, energy-efficient, and rich data source, aligned with the vision-centric design of real-world, human-designed environments.* Classical vision-based techniques such as Visual-inertial odometry (VIO) [222], Simultaneous localization and mapping (SLAM) [227] or Structure from motion (SfM) [226] struggle with the ill-posed nature of image processing. They cannot leverage semantic priors to resolve ambiguous situations and require extensive calibration procedures. Furthermore, monocular solutions lack the ability to estimate scale. These challenges are exacerbated in multi-robot scenarios that require fast and accurate relative pose estimates for tasks like flocking [68], path planning [70], and collaborative perception [93]. Explicit agent detection [223–225, 228, 229] requires line-of-sight and is not platform-agnostic, whereas deep pose predictors [230–236] show promise but have are not widely adopted in MRS due to hardware constraints. Additionally, incorporating *uncertainty metrics* is essential for assessing the reliability of predictions in these applications.

[68]: Reynolds (1987), *Flocks, Herds and Schools: A Distributed Behavioral Model*
[70]: Sharon et al. (2015), *Conflict-based search for optimal multi-agent pathfinding*
[93]: Zhou et al. (2022), *Multi-Robot Collaborative Perception With Graph Neural Networks*

We introduce CoViS-Net, a decentralised, platform-agnostic visual spatial foundation model designed for real-world multi-robot applications. While foundation models are commonly trained *at scale* and *unsupervised* (resulting in *emergence*) [138], our model is trained on the HM3D indoor dataset in a *supervised* manner. We classify it as a foundation model nonetheless due to its *homogenisation* capabilities, allowing a wide range of applications [138] in the multi-robot domain. *This model operates in*

[138]: Bommasani et al. (2021), *On the Opportunities and Risks of Foundation Models*

real-time on onboard computers without the need for pre-existing network infrastructure, thus addressing critical scalability, flexibility, and robustness challenges. We design a deep distributed stateless architecture to tackle multi-robot uncertainty-aware relative pose estimation and Bird's-eye-view (BEV) representation prediction for regions without local camera coverage. We demonstrate our approach in real-world applications, including a multi-robot control task outside the laboratory. Figure 7.1 shows a snapshot from this deployment.

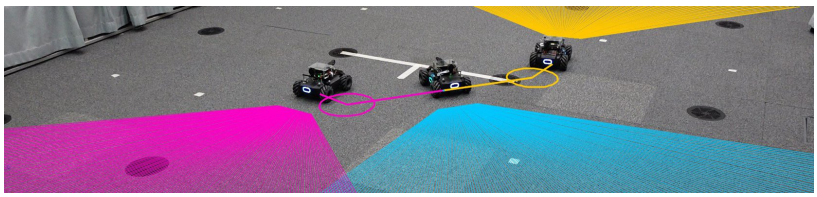


Figure 7.1: Our model can be used to control the relative pose of multiple follower robots (depicted in yellow and magenta) to a leader robot (blue) following a reference trajectory using visual cues in the environment only (the field-of-view is depicted by cones). The model can estimate the relative pose even if there is no direct visual overlap (cone intersection).

7.1 Contributions

Supplementary material

We provide the following supplementary material:

Website: Talk: Video:

Our contributions include the following:

- We introduce a novel architecture for platform-agnostic, decentralised, real-time multi-robot pose estimation from monocular images, incorporating uncertainty awareness without a predefined map. Estimating relative poses is a foundational ability for a wide range of multi-robot tasks, and our model allows this with a minimal amount of data exchanged between robots without any time-dependency (instantaneously).
- We extend it to predict uncertainty-aware BEV representations, facilitating spatial understanding in occluded areas. Similarly, this task is a key enabler for many real-world multi-robot tasks, and allows merging information observed from multiple robots for improved coordination.
- We validate our model in the real world for a multi-robot control task and show its capabilities even in ill-posed situations where images share no overlap.

7.2 Approach

In this section, we elaborate on the architecture and training pipeline of our foundation model. We detail our simulation dataset and define our model architecture before introducing the uncertainty estimation strategy and training protocol.

Given an MRS, our goals are: *(i)* for each robot to predict its pose and uncertainty relative to other robots as well as corresponding embeddings

using non-line-of-sight visual correspondences, and *(ii)* to use these embeddings for downstream tasks like local occupancy grid mapping or real-time multi-robot control in transfer learning [237]. Consider an MRS represented by a set of nodes $\mathcal{V}$. Each node $v_i \in \mathcal{V}$ has a position $\mathbf{p}^{w,i}$ and an orientation $\mathbf{R}^{w,i}$ in the world coordinate frame $\mathcal{F}_w$. The set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ defines the communication topology, connecting nodes v_i to v_j and defining the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. For each edge $(v_i, v_j) \in \mathcal{E}$, the relative position of v_j in the coordinate frame of v_i (denoted as $\mathcal{F}_i$) is $\mathbf{p}^{i,ij} = \mathbf{R}^{-1,w,i} \cdot (\mathbf{p}^{w,j} - \mathbf{p}^{w,i})$, and the relative rotation is $\mathbf{R}^{i,ij} = \mathbf{R}^{-1,w,i} \cdot \mathbf{R}^{w,j}$. Each node v_i is equipped with a camera C^i. The camera output at node v_i is represented as an image z^i. From the images, we estimate relative poses and associated uncertainties $(\hat{\mathbf{p}}^{i,ij},\ \sigma_{\mathrm{p}}^{2,ij},\ \hat{\mathbf{R}}^{i,ij},\ \sigma_{\mathrm{R}}^{2,ij})\ \forall\, v_j \in \mathcal{N}(v_i)$ with respect to v_i in its coordinate frame $\mathcal{F}_i$. Since we do not consider temporal data from previous time steps, we omit indexing variables at time t and assume that all data is processed at every time step ($z^i = z_t^i$). This approach does not assume overlap between images or line-of-sight visibility between robots, relying exclusively on the *relationship between image features*. This makes it platform-agnostic and requires only a single monocular camera. The robots cooperatively generate a bird's eye view representation $\hat{\mathrm{BEV}}^i$ around each node v_i using these estimated poses, which can be used for various downstream multi-robot applications.

[237]: Weiss et al. (2016), *A survey of transfer learning*

7.2.1 Model

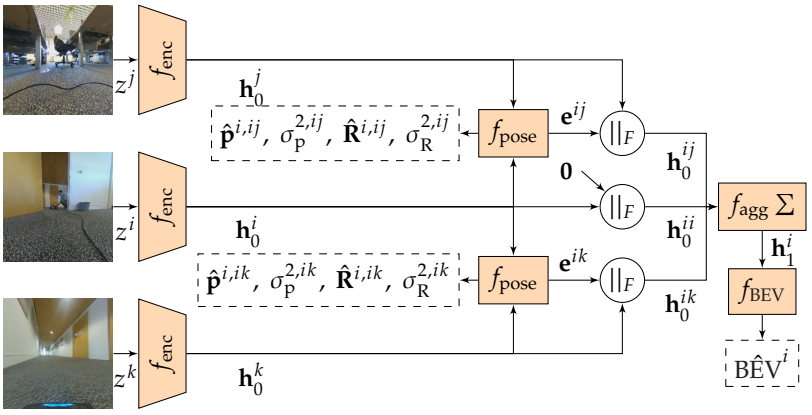


Figure 7.2: Overview of the model architecture: We illustrate the decentralised evaluation with respect to ego-robot v_i, which receives embeddings $\mathbf{h}_0^j, \mathbf{h}_0^k$ generated by the encoder f_{enc} from the neighbours $v_j, v_k \in \mathcal{N}(v_i)$. For each received embedding, we employ f_{pose} to compute the pose. We concatenate the pose embedding $\mathbf{e}^{ij}$ with the image embedding $\mathbf{h}_0^j$ (and $\mathbf{e}^{ik}$ with $\mathbf{h}_0^k$) and subsequently aggregate this into a node feature $\mathbf{h}_1^i$. Model outputs are framed with a dashed line.

We employ four primary models: The encoder f_{enc} generates a common embedding of images, f_{pose} estimates pairwise poses, f_{agg} aggregates node features, and f_{BEV} predicts a two-dimensional BEV representation. An overview of the model architecture is provided in Figure 7.2. Our model utilizes the Transformer architecture [139, 238] in f_{enc}, f_{pose} and f_{agg}. Self-attention computes the output as a weighted sum of the values ($\mathbf{V}$), with the weights determined by the compatibility of queries ($\mathbf{Q}$) and keys ($\mathbf{K}$) as per the formula: $\mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}((\mathbf{Q}\mathbf{K}^T)/\sqrt{d_k})\mathbf{V}$, where d_k is the dimensionality of the keys. Typically, $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are set to the same input features.

[139]: Vaswani et al. (2017), *Attention is All you Need*

[238]: Dosovitskiy et al. (2021), *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*

DinoV2 is a vision foundation model based on the vision transformer (ViT) and pre-trained in a self-supervised manner on the LVD-142M dataset, which is a cumulation of various filtered datasets, including various versions of ImageNet [239] and Google Landmarks.

[239]: Deng et al. (2009), *ImageNet: A large-scale hierarchical image database*

Image encoder The encoder f_{enc} consists of the pre-trained DinoV2 vision transformer and two more transformer layers previously described as $f_{\text{trans}}^{\text{enc}}$. We use the smallest distilled version of DinoV2 based on ViT-S/14 with 22.1 M parameters for performance reasons. The input feature size is 224×224, of which DinoV2 generates an embedding of size $S = 256$ and $F = 384$. We add a linear layer that transforms F down to the parameter described in this chapter (e.g. 24) while maintaining S. The output of this is then cropped to the S described in this chapter (e.g. 128) by cropping the sequence length at that position.

$$f_{\text{enc}} = f_{\text{dino}} \cdot f_{\text{trans}}^{\text{enc}} \tag{7.1}$$

$$f_{\text{enc}}(z^i) = \mathbf{h}_0^i \tag{7.2}$$

The weights in f_{dino} are frozen, and an instance of $f_{\text{trans}}^{\text{enc}}$, which is trained, is concatenated, resulting in f_{enc} (Equation 7.2).

Each robot v_i processes the image z^i through f_{enc} to generate the communicated node embedding $\mathbf{h}_0^i \in \mathbb{R}^{S \times F}$, where S represents the sequence length, and F the feature vector size. We freeze the weights of DinoV2, enabling the use of various open-source downstream fine-tuned image detectors without an additional forward pass.

Pairwise pose encoder For each ego-robot (i.e., root node v_i), the image encoding $\mathbf{h}_0^i$ is transmitted to neighbouring robots. Given encodings $\mathbf{h}_0^i$ and $\mathbf{h}_0^j$, f_{pose} estimates relative poses and uncertainties $(\hat{\mathbf{p}}^{i,ij},\ \sigma_{\text{p}}^{2,ij},\ \hat{\mathbf{R}}^{i,ij},\ \sigma_{\text{R}}^{2,ij})$ and edge embeddings $\mathbf{e}^{ij}$.

We define the concatenation operation of two tensors $\mathbf{a}$ and $\mathbf{b}$ along axis c as $(\mathbf{a} \mid\mid_c \mathbf{b})$ and the indexing operation on tensor $\mathbf{a}$ by the first element along axis b as $\mathbf{a}_{b1}$. We introduce an instance of another transformer $f_{\text{trans}}^{\text{pose}}$, and four MLPs f_{p}^{μ}, f_{p}^{σ}, f_{R}^{μ}, f_{R}^{σ}. Furthermore, we introduce a learnable transformer positional embedding for the sequence dimension $\mathbf{S}_{\text{pose}}$. For each incoming $\mathbf{h}_0^j \; \forall \, v_j \in \mathcal{N}(v_i)$, we perform a pose estimation as

$$\mathbf{e}^{ij} = f_{\text{trans}}^{\text{pose}}\left(\left(\mathbf{h}_0^i \mid\mid_S \mathbf{h}_0^j\right) + \mathbf{S}_{\text{pose}}\right)_{S1} \tag{7.3}$$

$$\hat{\mathbf{p}}^{i,ij},\ \hat{\mathbf{R}}^{i,ij} = f_{\text{p}}^{\mu}(\mathbf{e}^{ij}),\ f_{\text{R}}^{\mu}(\mathbf{e}^{ij}) \tag{7.4}$$

$$\sigma_{\text{p}}^{2,ij},\ \sigma_{\text{R}}^{2,ij} = f_{\text{p}}^{\sigma}(\mathbf{e}^{ij}),\ f_{\text{R}}^{\sigma}(\mathbf{e}^{ij}) \tag{7.5}$$

$$f_{\text{pose}}(\mathbf{h}_0^i, \mathbf{h}_0^j) = (\hat{\mathbf{p}}^{i,ij}, \hat{\mathbf{R}}^{i,ij}, \sigma_{\text{p}}^{2,ij}, \sigma_{\text{R}}^{2,ij}).$$

We concatenate the node features along the sequence dimension, add the positional embedding, run the transformer and then extract the first element of the sequence dimension (Equation 7.3), and eventually use this edge embedding to estimate poses and orientations (Equation 7.4) and uncertainties (Equation 7.5). We first project the embedding $\mathbf{h}_0^i$ to a larger feature size $F = 192$ using a linear layer. Each transformer layer has $F = 192$ in and out features. The learnable transformer positional embedding $\mathbf{S}_{\text{pose}}$ is initialised to match F and S, with normal distribution initialisation of $\sigma = 0.02$. The MLPs f_{P}^{μ}, f_{P}^{σ}, f_{R}^{μ}, f_{R}^{σ} are represented by one linear layer mapping the output of the transformer from $F = 192$ to $F = 17$ (position: 3, position uncertainty: 3, orientation: 10, orientation uncertainty: 1). We then use this edge embedding to estimate position, rotation and uncertainties, resulting in $f_{\text{pose}}(\mathbf{h}_0^i, \mathbf{h}_0^j) = (\hat{\mathbf{p}}^{i,ij},\ \sigma_{\text{P}}^{2,ij},\ \hat{\mathbf{R}}^{i,ij},\ \sigma_{\text{R}}^{2,ij})$.

Multi-node aggregation We perform the position-aware feature aggregation, utilising the node embeddings $\mathbf{h}_0^i$ generated in f_{pose} for each corresponding edge. The feature aggregation function f_{agg} integrates information from the set of nodes $v_j \in \mathcal{N}(v_i) \cup \{v_i\}$. The resulting output is a feature vector $\mathbf{h}_1^i$ for each node v_i, encapsulating the aggregated information from its local vicinity, which can be used for any downstream task in a transfer-learning context [237], requiring geometric information in a robotic swarm, for example, to predict a BEV of the environment around the robot that captures areas that might not necessarily be visible by the robot itself.

[237]: Weiss et al. (2016), *A survey of transfer learning*

We introduce an instance of two more transformers $f_{\text{trans}}^{\text{agg}}$ and $f_{\text{trans}}^{\text{post}}$, an MLP f_{aggpos}, a learnable transformer positional embedding $\mathbf{S}_{\text{agg}}$, as well as a BEV decoder f_{BEV}.

The feature aggregation function f_{agg} is built from $f_{\text{trans}}^{\text{agg}}$, which consists of five transformer layers, and $f_{\text{trans}}^{\text{post}}$, which consists of one transformer layer, all of which have an embedding size of $F = 192$. The MLP f_{aggpos} consists of three layers with 17, 48 and 48 neurons each, the positional embedding $\mathbf{S}_{\text{agg}}$ is initialised similar to the one in f_{pose}, and the BEV decoder f_{BEV} takes the first sequence element of $f_{\text{trans}}^{\text{post}}$ as input and scales it up through seven alternations of convolutions and upsampling.

$$\mathbf{x}^{ij} = \left(\left(\mathbf{h}_0^i \,||_S\, \mathbf{h}_0^j\right) + \mathbf{S}_{\text{agg}}\right) \tag{7.6}$$

$$\mathbf{h}_0^{ij} = \left(\mathbf{x}^{ij} \,||_F\, f_{\text{aggpos}}(\mathbf{e}^{ij})\right) \tag{7.7}$$

$$\mathbf{h}_0^{ii} = \left(\mathbf{x}^{ii} \,||_F\, f_{\text{aggpos}}(\mathbf{0})\right) \tag{7.8}$$

$$\mathbf{h}_1^i = f_{\text{trans}}^{\text{post}}\left(\sum_{v_j \in \mathcal{N}(v_i)\cup\{v_i\}} f_{\text{trans}}^{\text{aggr}}(\mathbf{f}_{ij})\right)_{S1} \tag{7.9}$$

$$\hat{\text{BEV}}^i = f_{\text{BEV}}(\mathbf{h}_1^i). \tag{7.10}$$

We concatenate the node features along the sequence dimension and add the positional embedding (Equation 7.6). We then concatenate a nonlinear transformation of the edge features along the feature dimension to the output of the previous operation (Equation 7.7). The previous operations are performed for all $v_j \in \mathcal{N}(v_i)$, but it is essential to include self-loops to the graph topology, as a full forward pass of the neural network would otherwise not be possible if $\mathcal{N}(v_i)$ were empty. We, therefore, set the edge embedding to the zero tensor $\mathbf{0}$ in that case (Equation 7.7). We then aggregate over the set of neighbours and itself to produce the output embedding (Equation 7.9) and finally use this to estimate any other downstream task, for example, estimating a BEV (Equation 7.10).

More generally, it is clear how this formulation integrates into the general GNN message passing formulation from Section 2.3.2 as

$$\mathbf{h}_{k+1}^{i} = \phi_{\text{post}}\left(\mathbf{h}_{k}^{i}, \sum_{v_j \in \mathcal{N}(v_i) \cup \{v_i\}} \phi_{\text{pre}}(\mathbf{h}_{k}^{i}, \mathbf{h}_{k}^{j}, \mathbf{e}^{ij})\right)$$

with $K = 1$ layers.

All transformer blocks have 12 attention heads. The MLP used in the transformer has a hidden size of $4F$. All blocks utilise a dropout of 0.2.

7.2.2 Training

For training our models, we use supervised learning. This section describes the dataset, loss functions, and uncertainty estimation.

[19]: Savva et al. (2019), *Habitat: A Platform for Embodied AI Research*
[240]: Szot et al. (2021), *Habitat 2.0: Training Home Assistants to Rearrange their Habitat*
[241]: Ramakrishnan et al. (2021), *Habitat-Matterport 3D Dataset (HM3D): 1000 Large-scale 3D Environments for Embodied AI*

Dataset We utilise the Habitat simulator [19, 240] along with the training set of the HM3D dataset [241], which comprises 800 scenes of 3D-scanned real-world multi-floor buildings. We select Habitat and HM3D for their photorealism and the availability of ground-truth pose information, as well as detailed navigation mesh information that can be converted to a BEV representation of the scene. We instantiate a single agent equipped with a calibrated camera with a field of view of 120°. For each scene, we randomly sample several uniform agent positions based on the total area of the scene. For each sample, the camera orientation is randomised across all three axes. Additionally, for each scene, we extract the processed navigation mesh, which indicates areas that are navigable by a mobile robot, or not navigable due to obstacles or overhangs. We extract 3,816,288 images from a calibrated camera with a 120° field of view from 800 scenes and 1,411 floors, along with corresponding BEV representations.

The samples of the datasets $\mathcal{D}^{\text{Sim}}_{\text{Train}}$ $\mathcal{D}^{\text{Sim}}_{\text{Test}}$ and $\mathcal{D}^{\text{Sim}}_{\text{Val}}$ are randomly sampled to be within orientation range $[-\pi, \pi]$ for roll and pitch, and within

a height of 0.0 to 2.0 meters, or less if the ceiling is at a lower height, resulting in uniformly distributed 6D poses.

We divide our data into training $\mathbb{D}^{\text{Sim}}_{\text{Train}}$ (80%), testing $\mathbb{D}^{\text{Sim}}_{\text{Test}}$ (19%), and validation $\mathbb{D}^{\text{Sim}}_{\text{Val}}$ (1%) sets. From these sets, we sample pairs of $N_{\text{max}} = 5$ images and poses by (1) uniformly sampling 25% of individual images from each dataset, and then (2) uniformly sampling $N_{\text{max}} - 1$ images within a distance of $d_{\text{max}} = 2\,\text{m}$ around each sample from (1). This process yields 763, 257 training and 276, 680 test tuples, each of which is at most $2d_{\text{max}}$ apart and pointing in random directions, thus providing realistically close and distant data samples for estimating both poses and uncertainties in case of no overlaps.

The ground truth BEV BEV^i is a cropped 6 m × 6 m view of the floor's BEV around v_i, generated through Habitat's navigation mesh. Each agent v_i is centred and facing north.

Uncertainty estimation Two types of uncertainties can be modelled: *epistemic uncertainty*, which captures uncertainty in the model and can be reduced with more data, and *aleatoric uncertainty*, which captures noise inherent to specific observations [242]. For pose estimation and pose control in robotic systems, aleatoric uncertainty is more important since downstream applications can *act* on it to *reduce* it. Traditional methods like Monte Carlo Dropout [243] and Model Ensembles [244] model epistemic uncertainty. Instead, we propose using the Gaussian negative log likelihood (GNLL) loss [242, 245], a generalisation of the mean squared error loss, allowing the model to predict both a mean $\hat{\mu}$ and a variance $\hat{\sigma}^2$, learned from data points μ. The loss is defined as

$$L^{\text{GNLL}}\left(\mu, \left(\hat{\mu}, \hat{\sigma}^2\right)\right) = \frac{1}{2}\left(\log\left(\hat{\sigma}^2\right) + \frac{(\hat{\mu} - \mu)^2}{\hat{\sigma}^2}\right). \tag{7.11}$$

The GNLL loss penalises high variance predictions and scales the mean squared error loss by the predicted variance, penalising errors more when the predicted uncertainty is high.

[242]: Kendall et al. (2017), *What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?*

[243]: Gal et al. (2016), *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*

[244]: Lakshminarayanan et al. (2017), *Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles*

[242]: Kendall et al. (2017), *What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?*

[245]: Nix et al. (1994), *Estimating the mean and variance of the target probability distribution*

Estimating rotations Estimating rotation is difficult due to discontinuities in many standard orientation representations. Quaternions are often used for their numerical efficiency and lack of singularities. Related work [246] represents rotations with a symmetric matrix defining a Bingham distribution over unit quaternions to predict epistemic uncertainty. We combine the chordal loss introduced in that work and GNLL loss to predict aleatoric uncertainty for rotations represented as quaternions, as

[246]: Peretroukhin et al. (2020), *A Smooth Representation of Belief over SO(3) for Deep Rotation Learning with Uncertainty*

shown in the following equation:

$$d_{\text{quat}}(\hat{\mathbf{q}}, \mathbf{q}) = \min\left(\|\mathbf{q} - \hat{\mathbf{q}}\|_2, \|\mathbf{q} + \hat{\mathbf{q}}\|_2\right) \tag{7.12}$$

$$L^2_{\text{chord}}(\hat{\mathbf{q}}, \mathbf{q}) = 2d^2_{\text{quat}}(\hat{\mathbf{q}}, \mathbf{q})\left(4 - d^2_{\text{quat}}(\hat{\mathbf{q}}, \mathbf{q})\right) \tag{7.13}$$

$$L^{\text{GNLL}}_{\text{chord}}(\mathbf{q}, (\hat{\mathbf{q}}, \hat{\sigma}^2)) = \frac{1}{2}\left(\log(\hat{\sigma}^2) + \frac{L^2_{\text{chord}}(\hat{\mathbf{q}}, \mathbf{q})}{\hat{\sigma}^2}\right). \tag{7.14}$$

The model learns a ten-dimensional symmetric matrix that is mapped to quaternions, and Equation 7.12 in particular ensures continuity between the ambiguous quaternion representations.

[247]: Sudre et al. (2017), *Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations*
[248]: Jadon (2020), *A survey of loss functions for semantic segmentation*

Training We use a combination of the Dice loss [247] L_{Dice} and Binary cross entropy (BCE) Loss L_{BEC} weighted by the parameter $0 \leq \alpha \leq 1$. This is commonly referred to as *combo loss* [248] and is typically utilised for slight class imbalance. We define the total loss as

$$L_{i,\text{BEV}} = \alpha \cdot L_{\text{Dice}}\left(\text{BEV}^i, \hat{\text{BEV}}^i\right) + (1-\alpha) \cdot L_{\text{BCE}}\left(\text{BEV}^i, \hat{\text{BEV}}^i\right) \tag{7.15}$$

$$L_{i,ij,\text{Pose}} = (1-\beta)\, L^{\text{GNLL}}\left(\mathbf{p}^{i,ij}, \left(\hat{\mathbf{p}}^{i,ij}, \sigma_{\text{p}}^{2,ij}\right)\right) + \beta\, L^{\text{GNLL}}_{\text{chord}}\left(\mathbf{R}^{i,ij}, \hat{\mathbf{R}}^{i,ij}, \sigma_{\text{R}}^{2,ij}\right) \tag{7.16}$$

$$L = \sum_{v_i \in \mathcal{V}}\left(L_{i,\text{BEV}} + \sum_{v_j \in \mathcal{N}(v_i)} L_{i,ij,\text{Pose}}\right). \tag{7.17}$$

The parameter $0 \leq \beta \leq 1$ balances between the position and orientation loss. Uncertainty terms are necessary for training, as many dataset samples lack overlap and may be in adjacent rooms, leading to significant errors from difficult samples that could hinder learning. We use Pytorch Lightning to train our model for 15 epochs on two NVidia A100 GPUs, using the simulated data $\mathcal{D}^{\text{Sim}}_{\text{Train}}$. This training takes approximately 24 hours. We optimise the weights using the AdamW optimiser and use the 1cycle learning rate annealing schedule throughout the training and configured weight decay. We choose the best model based on the performance of the validation set.

We set the initial learning rate to 1^{-5} and configure a learning rate schedule to increase this over two epochs up to 1^{-3}, followed by a sinusoidal decline to 0 over 20 epochs. The parameter α balances between Dice and BCE loss, and we set it to $\alpha = 0.5$, and the parameter β balances between position and orientation prediction accuracy, and we set it to 1, given equal weight to both.

7.3 Results

We demonstrate the performance of our trained models on the test set $\mathcal{D}_{\text{Test}}^{\text{Sim}}$ and custom real-world test sets $\mathcal{D}_{\text{Test}}^{\text{Real}}$. First, we outline the metrics used. Then, we introduce the real-world dataset. Next, we analyse model performance on both simulation and real-world data. Finally, we compile and deploy our model in a real-world, real-time trajectory tracking task.

Metrics We compute the Euclidean distance between the predicted and ground truth poses as $D_{\text{pos}}(\mathbf{p}^{i,ij}, \hat{\mathbf{p}}^{i,ij})$ and the geodesic distance between rotations as $D_{\text{rot}}(\mathbf{R}^{i,ij}, \hat{\mathbf{R}}^{i,ij})$ for all edges and report median errors. We categorise the median pose error into four categories. *All* includes all edges. *Visible* and *Invisible* edges are determined by FOV overlap (an edge is invisible if $D_{\text{rot}}(\mathbf{R}^{i,ij}, \mathbf{0}) >$ FOV). *Invisible Filtered* samples use the predicted position uncertainty $f_{\text{p}}^{\sigma} ij$ to reject high-uncertainty pose estimations based on the Youden's index [249]. For simulation results, we also report the Dice and Intersection over union (IoU) metrics. We compute the absolute Euclidean distance between the predicted and ground truth poses and the geodesic distance between rotations as

[249]: Youden (1950), *Index for rating diagnostic tests*

$$D_{\text{pos}}(\mathbf{p}^{i,ij}, \hat{\mathbf{p}}^{i,ij}) = \left\| \mathbf{p}^{i,ij} - \hat{\mathbf{p}}^{i,ij} \right\|$$
$$D_{\text{rot}}(\mathbf{R}^{i,ij}, \hat{\mathbf{R}}^{i,ij}) = 4 \cdot \arcsin\left(\frac{1}{2} d_{\text{quat}}(\hat{\mathbf{R}}^{i,ij}, \mathbf{R}^{i,ij}) \right)$$

for all edges and report median errors.

Stability We show the train and validation loss for $N = 5$ training runs with different seeds in Figure 7.3. Training with $F = 48$ and $S = 128$ across $N = 5$ different seeds reveals a low average standard deviation of 0.011374, leading us to report results for only $N = 1$ seeds in Table 7.1 and Table 7.3.

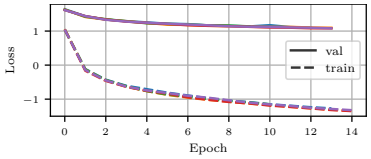


Figure 7.3: We show the training and validation loss for a model trained with $F = 48$ and $S = 128$ over $N = 5$ different training seeds. We report a low average standard deviation of 0.011374 over all epochs on the validation loss.

7.3.1 Simulation

We conducted experiments across various model variants trained with different parameters, reporting on our model's efficiency. We present quantitative results in Table 7.1 and Table 7.2, with qualitative findings in Figure 7.4. We provide additional qualitative samples in Section C.1.

Ablation study Table 7.1 details the performance across seven different experiments. We evaluated the impact of image embedding sizes $\mathbf{h}_0^i$ on communication efficiency. The IoU and Dice metrics for the BEV representation, alongside median pose estimates for all samples in the dataset, are provided. Experiments show that the sequence length S significantly

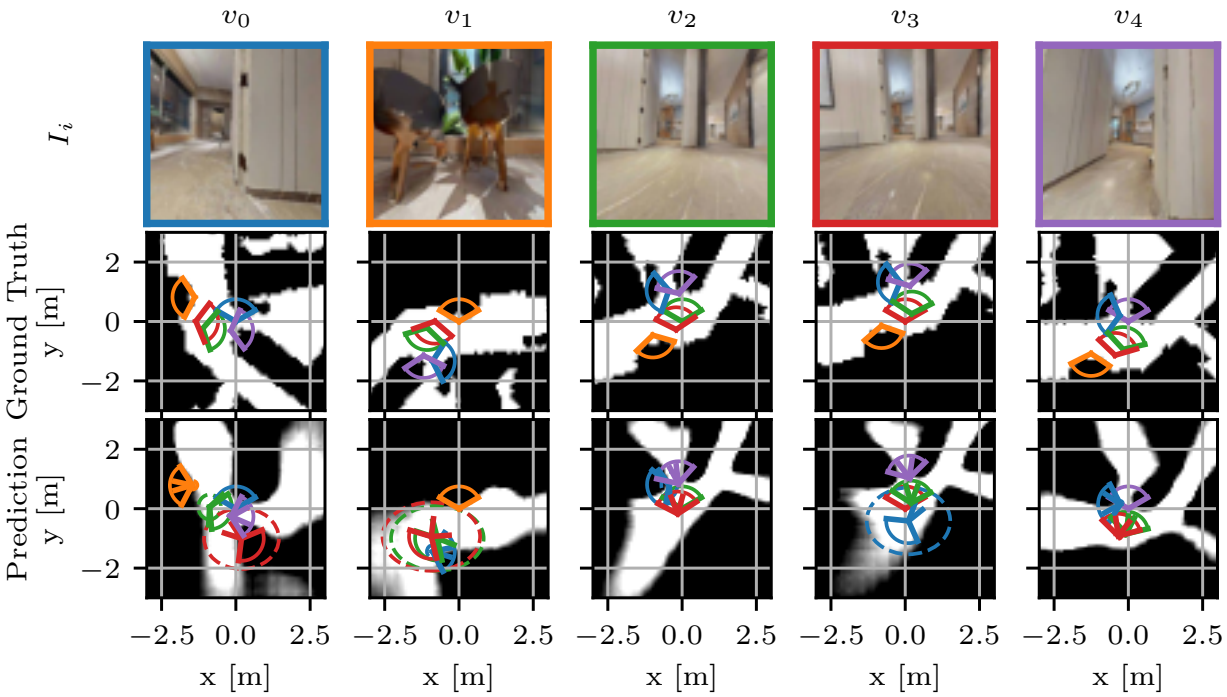


Figure 7.4: By leveraging priors, such as the way humans lay out their rooms, our model provides rough relative positions *even without overlapping fields of view*, which classical approaches cannot achieve. This figure shows a sample from the simulation validation dataset $\mathcal{D}_{\text{Val}}^{\text{Sim}}$ for five nodes, each represented by a column. The top row displays the camera image z^i from each node's perspective. The middle row shows the ground truth labels in the coordinate system $\mathcal{F}_i$, with each robot centred and facing upwards at (0, 0), and the ground truth poses $\mathbf{p}^{i,ij}$ of other robots indicated, along with their field of view. The background shows the BEV representation BEV^i for each robot. The bottom row presents the corresponding predictions, displaying pose predictions $\hat{\mathbf{p}}^{i,ij}$, $\hat{\mathbf{R}}^{i,ij}$ with low uncertainty $\sigma_{\text{p}}^{2,ij}$ $\sigma_{\text{R}}^{2,ij}$, predicted uncertainties $\sigma_{\text{p}}^{2,i}ij$ as dashed oval and $\sigma_{\text{R}}^{2,i}ij$ similar to the FOV, and the predicted BEV representation $\hat{\text{BEV}}^i$ in the background.

influences BEV representation and pose estimate performance more than feature size F. We set S to 1 for two experiments: *(1)* increasing F to match the size of the embeddings for other experiments, and *(2)* decreasing F, effectively reducing the transformer to a CNN-only architecture by removing the attention mechanism. Both experiments result in reduced performance, highlighting the importance of attention in our model. The best-performing model with $S = 256$ achieves an IoU of 57.1, a Dice score of 69.1, and median pose accuracy of 36 cm and 8.3°, which is roughly equal to one robot body length. In contrast, models with $S = 1$ show significant degradation, with nearly unusable BEV representation predictions and pose errors on the scale of meters. Models with $F = 24$ and $S = 64$ performed slightly worse than those with larger F or S.

BEV ablation We assess the effectiveness of BEV predictions, focusing on the contribution of pose predictions to enhancing BEV accuracy, in an ablation study detailed in Table 7.2.

Model			BEV		Median Pose Err.	
S	F	Mem	Dice	IoU		
256	48	24 kB	**69.1**	**57.1**	**36 cm**	8.3°
128	96	24 kB	68.8	56.8	40 cm	8.4°
128	48	12 kB	67.9	56.1	38 cm	**7.7°**
128	24	6 kB	66.7	54.6	50 cm	9.5°
64	48	6 kB	61.0	43.5	51 cm	9.6°
1	3072	6 kB	47.0	1.4	144 cm	89.9°
1	348	696 B	47.1	1.4	84 cm	11.7°

Table 7.1: Ablation study over the number of patches and size of features for each patch. We report the BEV performance as well as the median error for poses on the dataset $\mathcal{D}^{\text{Sim}}_{\text{Test}}$. The size of the embedding $\mathbf{h}^i_0$ is reported in bytes.

Experiment	Dice	IoU	Median Pose Error
None	62.8	49.5	N/A
Predicted	68.3	56.1	31 cm, 5.0°
Ground truth	74.3	63.2	0 cm, 0.0°

Table 7.2: Ablation over different modes for the BEV prediction on the simulation test set $\mathcal{D}^{\text{Sim}}_{\text{Test}}$.

We conduct three experiments: *(1)* A model trained without incorporating any pose prediction into the BEV aggregator. This model can estimate local information and, to a degree, information aggregated through neighbours. However, it must implicitly learn to predict the relative poses between observations to successfully merge the BEV representations from multiple agents. *(2)* A model trained as described in this chapter, integrating pose estimates with BEV predictions and aggregating this information from neighbour nodes. For this experiment, we use the model configuration with $F = 48$ and $S = 128$. *(3)* A BEV model trained with ground truth pose information from the simulation training set $\mathcal{D}^{\text{Sim}}_{\text{Train}}$. This model's performance is considered an upper limit, assuming perfect pose estimation. Evaluation of the simulation test set $\mathcal{D}^{\text{Sim}}_{\text{Test}}$ indicates our model performs between the two baselines. It significantly outperforms the pose-free model with an 8.75% improvement in accuracy, while the model using ground truth poses shows an 18.31% performance increase compared to the baseline without pose information. These results highlight the substantial benefit of incorporating pose predictions into BEV estimation, positioning our approach as a notably effective middle ground between the two extremes.

We investigate the efficiency of the uncertainty estimation as well as the distributions of pose errors more closely. Figure 7.5 illustrates the distributions of position and rotation errors, along with the predicted variances over the angle difference between robot pairs in the dataset $\mathcal{D}^{\text{Sim}}_{\text{Test}}$. An angle difference of 0° signifies robots facing the same direction, potentially resulting in significant image overlap, while an angle difference of 180° indicates opposite directions, with no overlap. The Field Of View (FOV) overlap threshold is highlighted in red, delineating errors with overlap (below the line) from those without (above the line). The analysis reveals that position errors remain consistent up to the FOV threshold, beyond which they escalate, as reflected in the predicted position variance. Conversely, rotation errors stay uniform across angle differences, with only a marginal increase for higher angles, though the predicted rotation variance notably rises. The horizontal marginals show the distribution of each individual plot while the vertical marginal shows the distribution of angle differences over the dataset. While the position error and variance increase over the threshold, the majority of

the predictions are still useful. The rotation error is consistent across angle differences while the uncertainty increases.

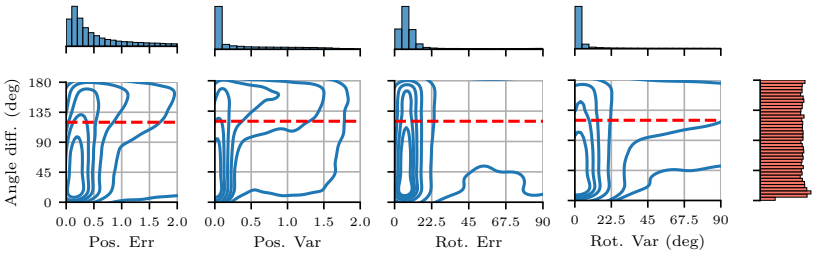


Figure 7.5: We show qualitative distributions for position and rotation errors and variances over the relative angle between two robots on all edges of the simulation test set $\mathcal{D}_{\text{Test}}^{\text{Sim}}$. We show the FOV as a dashed line, where any results below the line indicate that the edge has some image overlap and above no image overlap.

7.3.2 Real-world

In this section, we evaluate the model trained in Section 7.3.1 on real-world data. We detail our robot setup, custom real-world test set, and model evaluation on this dataset. We explain our real-world data collection setup in Figure 7.6 and Figure 7.7. We run a series of experiments with quantitative results in Table 7.3, Table 7.4, Table 7.5, Table 7.6, and Figure 7.8 and qualitative findings in Figure 7.9. We provide additional qualitative samples in Section C.2.

Real-world dataset Our setup includes four Cambridge RoboMasters [8] and one Unitree Go1 Quadruped, each equipped with local compute, a monocular camera and WiFi dongles for ad-hoc networking. We provide additional details on the camera calibration in Section A.3.2. Our custom real-world test dataset $\mathcal{D}_{\text{Test}}^{\text{Real}}$ covers a diverse and challenging range of indoor and outdoor scenes. Data collection involves a base station with dual cameras and a 2D Lidar, plus a distinctively shaped marker on each robot. We manually navigated robots around the base station to capture a variety of relative poses, then used a custom tool to label Lidar data for pose extraction.

[8]: Blumenkamp et al. (2024), *The Cambridge RoboMaster: An Agile Multi-Robot Research Platform*

All robots are equipped with an NVidia Jetson Orin NX 16GB and a forward-facing camera, rectified to a 120° FOV with an image resolution of 224×224 pixels. We collected the real-world test set $\mathcal{D}_{\text{Test}}^{\text{Real}}$ in eight indoor scenes. It consists of 5692 images. We sample image pairs to ensure a uniform distribution of relative positions from 0 m to 2 m. This results in 14008 sets of samples, each with three images, totalling 84048 pose edges, 32%, which have no visual overlap. Our dataset covers a diverse and challenging range of indoor scenes, as well as one outdoor scene.

We explain the setup of our data collection unit in Figure 7.6. We visualise the five different scenes we collected the dataset from as well as the data distribution from each respective scene in Figure 7.7.

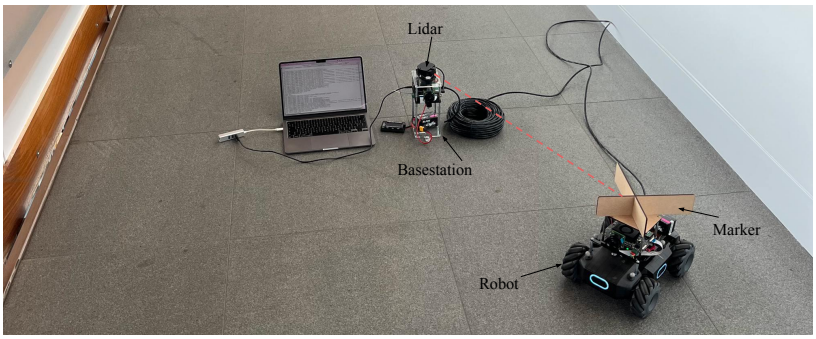


Figure 7.6: To collect the real-world test-set, we construct a static base station equipped with two cameras facing in opposite directions and a lidar. We add a marker on the robot so that its pose can be tracked with the lidar and then manually move the robot around the base station.

Ablation Evaluation on the real-world dataset $\mathcal{D}_{\text{Test}}^{\text{Real}}$ focuses on position and rotation accuracy, excluding quantitative BEV performance due to unavailable ground truth data. We first report results similar to Table 7.1 on $\mathcal{D}_{\text{Test}}^{\text{Real}}$ in Table 7.3. The results align with $\mathcal{D}_{\text{Test}}^{\text{Sim}}$, showing a correlation between the number of patches S and features F and pose estimation performance. The rotation estimates do not significantly differ between visible and invisible samples, demonstrating robust pose estimation without requiring visual overlap. The most accurate model achieves a median localisation error of 97 cm and 7.9° for invisible samples, and 33 cm and 5.8° for visible samples. Note that $\mathcal{D}_{\text{Test}}^{\text{Real}}$ samples are at most 2 m apart, resulting in a worst-case error of up to 4 m. These findings demonstrate our model's capability to reliably estimate poses, even with minimal overlap, and accurately predict higher variances under adverse conditions. We attribute this performance to our novel transformer-based pose prediction architecture, as well as the robust spatial representation capabilities of the DinoV2 encoder. The model goes beyond mere feature matching, achieving sophisticated scene understanding that allows it to "imagine" unseen portions of the scene based on learned data priors, thus facilitating pose estimation.

S	F	Mem	Invisible Filtered		Invisible		Visible	
256	48	24 kB	61 cm	**6.8°**	**97 cm**	7.9°	33 cm	5.8°
128	96	24 kB	**55 cm**	7.7°	97 cm	**7.4°**	32 cm	**5.6°**
128	48	12 kB	67 cm	9.9°	113 cm	9.6°	**29 cm**	5.7°
128	24	6 kB	83 cm	11.2°	112 cm	9.7°	31 cm	5.7°
64	48	6 kB	81 cm	9.4°	119 cm	10.8°	36 cm	6.3°
1	3072	6 kB	123 cm	25.7°	122 cm	25.8°	93 cm	138.2°
1	348	696 B	150 cm	164.1°	135 cm	37.9°	80 cm	11.1°

Table 7.3: Ablation study over the number of patches and size of features for each patch. We report the BEV performance as well as the median error for poses on the dataset $\mathcal{D}_{\text{Test}}^{\text{Real}}$. The size of the embedding $\mathbf{h}_0^i$ is reported in bytes.

Baseline We conduct a baseline comparison with two feature-matching-based approaches, as summarized in Table 7.4. Both baseline methods extract and match features across two images to estimate the essential matrix, deriving rotation and scale-less translation vectors. The first baseline uses OpenCV for ORB feature extraction followed by brute-force matching. The second baseline, LightGlue, uses a neural network-based approach for feature extraction and matching. Given both baselines

Refer `https://docs.opencv.org/4.x/d3/da1/classcv_1_1BFMatcher.html` for details.

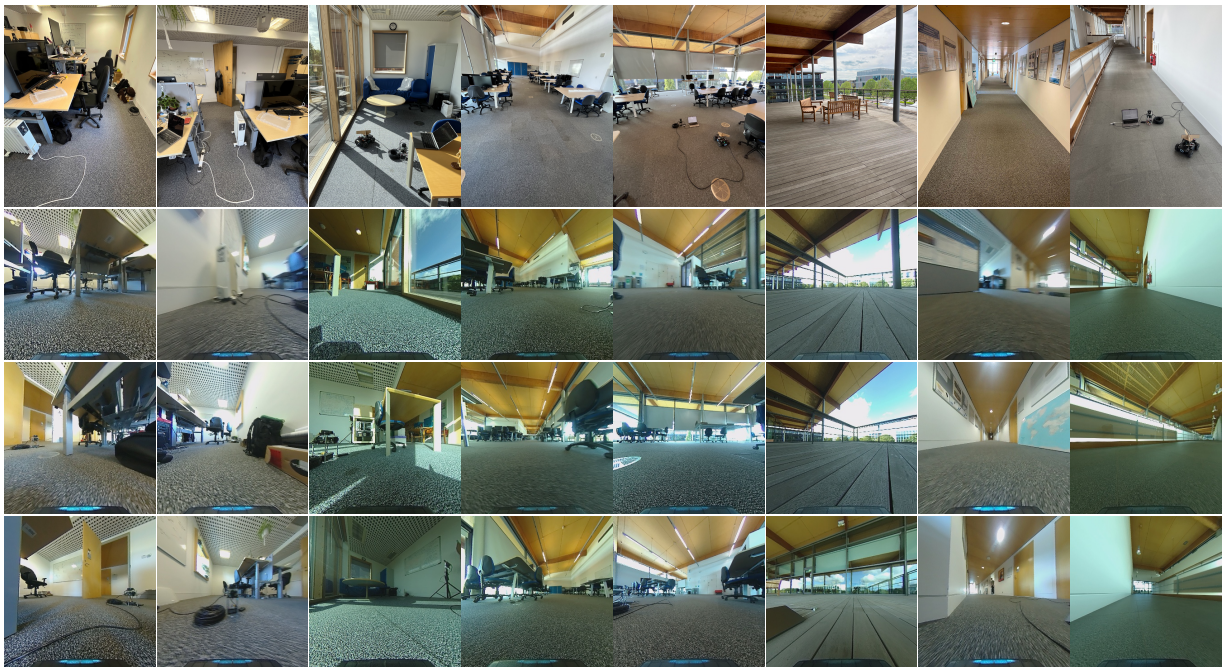


Figure 7.7: We collect the real-world dataset $\mathcal{D}_{\text{Test}}^{\text{Real}}$ from eight unique scenes, showing different challenges, from cluttered environments over scenes with high ceilings to sunny floors. The first row shows a picture summary of the scene, taken from a first-person-view perspective. The second to fourth row shows three samples from the real-world dataset, from the perspective of the robot. We show additional data in Figure C.9.

[250]: Lindenberger et al. (2023), *LightGlue: Local Feature Matching at Light Speed*

estimate scale-less transformations, we adopt the same metric used in LightGlue [250], computing the AUC at thresholds of 20, 45, and 90 degrees based on the maximum error in rotation and translation. Our findings indicate that, while both baselines require visual overlap to function effectively, our approach does not, highlighting its robustness and applicability in a broader range of scenarios.

Table 7.4: We report the AUC metric at 20, 45, and 90°on a first-principles and an NN-based baseline.

	All			Invisible			Visible		
AUC	20	45	90	20	45	90	20	45	90
ORB/OpenCV [251]	6.45	15.43	28.26	0.16	1.60	7.43	9.41	21.92	38.03
LightGlue [250]	16.30	27.14	37.63	0.02	0.09	0.31	23.94	39.82	55.14
Ours	**25.23**	**47.63**	**66.34**	**9.22**	**24.40**	**45.74**	**32.74**	**58.53**	**76.01**

Per scene In Table 7.5, we investigate the performance of our model across the different scenes in the testing dataset $\mathcal{D}_{\text{Test}}^{\text{Real}}$. We provide a detailed breakdown of the median pose error for Invisible, Invisible Filtered and Visible cases for each of the eight scenes. While the pose error for visible samples is consistently between 22 cm/5.2°and 48 cm/7.5°, there are more outliers for invisible samples. This is due to some scenes having more or less symmetry, thus making it more challenging for the model to accurately predict poses. The pose prediction for the outdoor scene are least accurate, which is to be expected since the model was trained on indoor scenes. The Sunny scene has severe shadows and direct sunlight, while still providing low pose errors. The pose predictions are useful in all scenarios, demonstrating impressive generalisation ability

across a wide range of scenes.

Scenario	Invisible		Invisible Filtered		Visible	
Corridor A	138 cm	129.6°	90 cm	11.4°	22 cm	5.2°
Corridor B	112 cm	**5.9°**	261 cm	8.2°	31 cm	5.2°
Office A	86 cm	14.4°	**38 cm**	9.6°	33 cm	7.5°
Office B	146 cm	10.4°	248 cm	**4.6°**	**20 cm**	**4.7°**
Outdoor	134 cm	9.5°	193 cm	28.7°	49 cm	6.9°
Study A	132 cm	6.7°	40 cm	5.1°	28 cm	5.3°
Study B	115 cm	8.9°	56 cm	9.6°	34 cm	5.5°
Sunny	**77 cm**	9.3°	49 cm	8.7°	22 cm	6.0°

Table 7.5: The median pose estimation error of our model on each of the eight $\mathcal{D}_{\text{Test}}^{\text{Real}}$ scenes.

Error distributions Next, we evaluate the magnitude of pose prediction errors over different thresholds of distances between image pairs in Table 7.6. We consider five bands of distances separated in 0.4 m chunks, ranging up to 2m. Note that the worst-case pose error is $2 \cdot d$ due to the three-dimensional pose predictions. While the error increases with an increasing distance between samples, the rotational error in particular stays low, and the position errors stay useful, ranging from 12 cm to 65 cm for visible samples, 34 cm to 134 cm for invisible filtered samples (samples that are not visible but for which he model predicts a high confidence) and 76 cm to 153 cm for all invisible samples.

d	Invisible		Invisible Filtered		Visible	
<0.4 m	**76 cm**	**7.6°**	**34 cm**	**6.7°**	**12 cm**	**4.9°**
<0.8 m	80 cm	8.8°	48 cm	8.6°	21 cm	5.2°
<1.2 m	95 cm	9.5°	56 cm	8.5°	31 cm	5.8°
<1.6 m	123 cm	10.6°	97 cm	12.9°	45 cm	6.0°
<2.0 m	153 cm	10.5°	134 cm	16.2°	65 cm	6.8°

Table 7.6: We show median pose prediction errors for different distance thresholds d on the dataset $\mathcal{D}_{\text{Test}}^{\text{Real}}$.

We perform a further evaluation on the distributions of pose errors and variations on the simulation test set $\mathcal{D}_{\text{Test}}^{\text{Real}}$ in Figure 7.8. The results are in line with the results performed on $\mathcal{D}_{\text{Test}}^{\text{Sim}}$ in Figure 7.5. The horizontal marginals show the distribution of each individual plot while the vertical marginal shows the distribution of angle differences over the testset. Note that the angle distribution is not uniform due to the real-world dataset collection where at least two cameras always face the opposite direction.

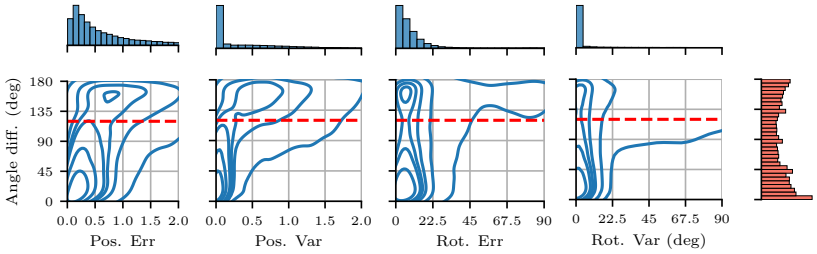


Figure 7.8: We show qualitative distributions for position and rotation errors and variances over the relative angle between two robots on all edges of the real-world testset $\mathcal{D}_{\text{Test}}^{\text{Real}}$, containing 84048 unique edges. We show the FOV as dashed line, where any results below the line indicate that the edge has some image overlap and above no image overlap.

Sample We provide a qualitative sample in Figure 7.9, where additionally to the simulation sample in Figure 7.4, we also visualise the prediction uncertainty. More samples can be found in the supplementary material.

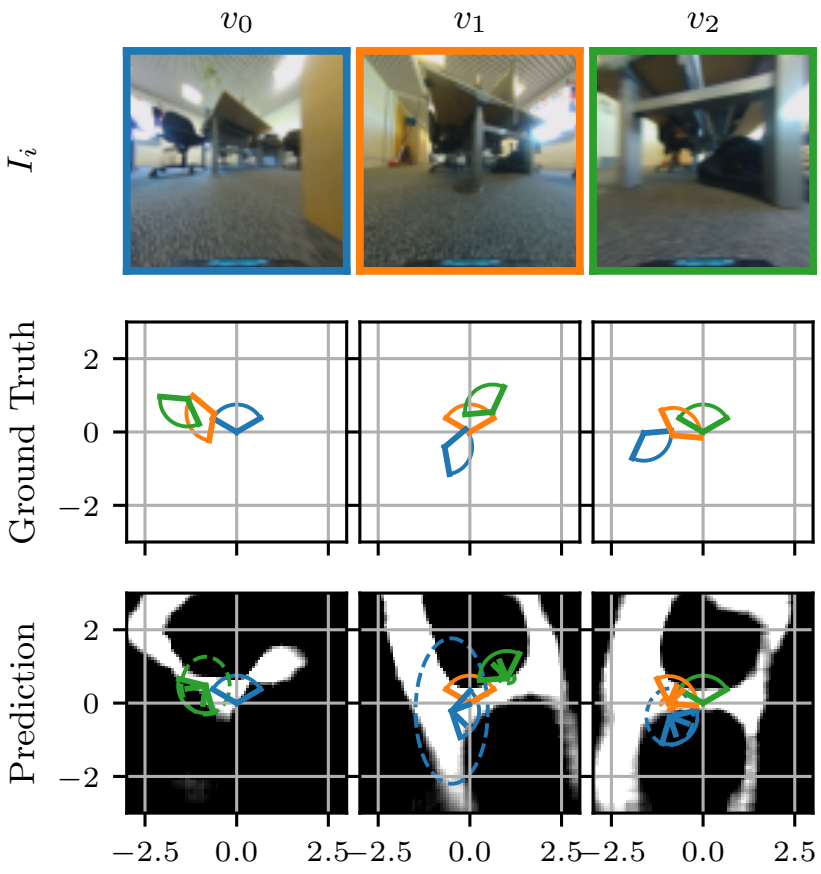


Figure 7.9: We visualise a sample from our custom real-world testset $\mathcal{D}_{\text{Test}}^{\text{Real}}$ for three nodes, similar to the simulation sample in Figure 7.4. We do not show the ground-truth BEV segmentation since we do not have labels for this. Additionally to the pose predictions $\hat{\mathbf{p}}^{i,ij}$, $\hat{\mathbf{R}}^{i,ij}$, we also visualise the predicted uncertainties $\sigma_{\text{p}}^{2,i}ij$ and $\sigma_{\text{R}}^{2,i}ij$.

7.3.3 Real-world pose control

In this section, we describe our experiment setup (including model deployment, wireless communication, and controllers), demonstrating real-world multi-robot control tasks. Results are reported in Figure 7.10, Table 7.7 and Figure 7.11. We provide additional material in Section C.3.

Model deployment We deploy a model with $F = 24$ and $S = 128$. After training in PyTorch, we compile f_{enc}, f_{pose}, and f_{agg} for decentralized deployment. Using TensorRT, we achieve sub-30 ms processing times. We provide more details and a runtime analysis in Section A.3.4.

Ad-hoc WiFi Image embeddings are sent between robots at 15 Hz, each over 6 KiB. The system's distributed nature suits broadcast communication, allowing scalable operation. However, IEEE 802.11/WiFi's fallback to low bit rates for broadcast increases frame loss probability, worsening with more robots. Higher-level systems like ROS2's CycloneDDS manage retransmissions but can't detect network overloads, leading to increased packet loss. We design a custom protocol that dynamically adjusts messaging load based on network conditions and uses TDMA scheduling to maximize frame reception without compromising data rates. We provide more details in Section A.3.5.

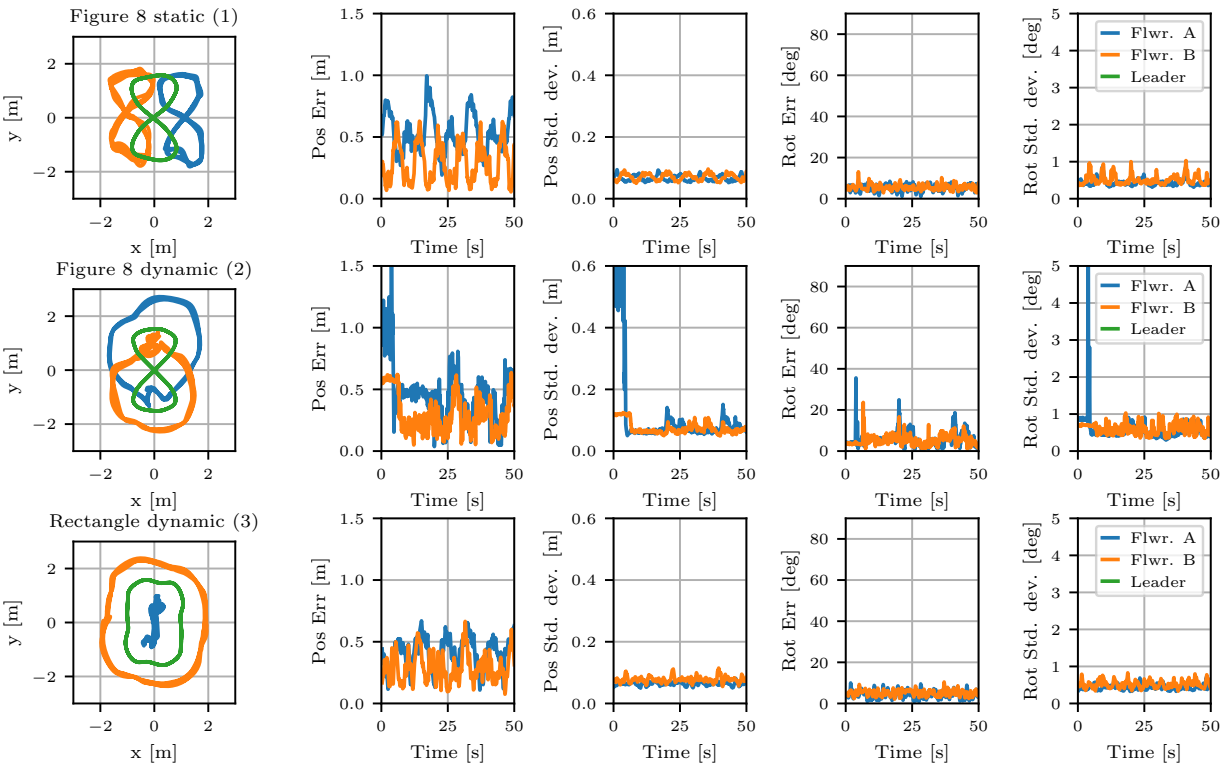


Figure 7.10: We evaluate the tracking performance of our model and uncertainty-aware controller on three reference trajectories, with two follower robots (in blue and orange), positioned left and right of the leader robot (in green). For each trajectory, we report the tracking performance over time for position and rotation, as well as the predicted uncertainties. The trajectory depicted in the first row (1) is a Figure Eight with the leader always facing in the same direction. The trajectory in the second row (2) is similar to the first, but the leader is always facing the direction it is moving. In this trajectory, Follower A starts facing the opposite direction of the leader, resulting in high position variance and low rotation variance, which decreases as the robot rotates, facing the same direction as the leader. A spurious erroneous rotation prediction is accompanied by a high uncertainty. Note that as the leader moves through the trajectory, there is always one inside and one outside follower robot. This role changes depending on the section of the figure, resulting in these follower trajectories. The trajectory depicted in the third row (3) is a rounded rectangle. We show the trajectory for 120 s and the tracking error for the first 60 s. The tracking performance is consistent across multiple runs.

Experiment We conduct qualitative experiments in various environments, including outdoors, and quantitative experiments. Two follower robots maintain a fixed orientation and distance from a leader robot on a predefined trajectory. A PD controller manages relative pose control, adjusting based on predicted uncertainty and selectively deactivating if uncertainty thresholds are exceeded. This ensures that even with unreliable estimates, robots maintain orientation toward the leader until reliable estimates are available. We assess tracking accuracy by comparing ground truth and estimated poses, focusing on position and rotation errors and predicted uncertainties.

Results Figure 7.11 shows snapshots from four different real-world deployments, and Figure 7.10 shows the trajectories of all robots for all quantitative experiments, as well as a visualisation of the position and rotation error over time. In the appendix, we provide additional snapshots of more real-world experiments.

The qualitative experiments show a robust system that generalises to a wide range of scenarios, including outdoors, albeit the scale prediction

is less reliable in outdoor settings. Our model was trained exclusively on indoor data and generalises to outdoor data due to the wide range of training data used on the DinoV2 encoder. The performance can be significantly improved by training on outdoor samples.

Table 7.7: We report the mean and absolute tracking error for both leaders for all three trajectories, as well as average velocities. The errors are consistent with the results reported in Table 7.3.

Trajectory	Robot	Mean Abs.	Median	Vel
Fig. 8 dyn.	A	38 cm, 5.6°	38 cm, 4.8°	0.59 m/s
	B	28 cm, 5.1°	25 cm, 4.7°	0.58 m/s
Fig. 8 stat.	A	51 cm, 5.2°	48 cm, 5.3°	0.60 m/s
	B	28 cm, 5.6°	26 cm, 5.4°	0.61 m/s
Rect. dyn.	A	41 cm, 4.4°	42 cm, 4.4°	0.32 m/s
	B	29 cm, 5.1°	27 cm, 5.1°	0.81 m/s

The tracking error in the quantitative experiments oscillates throughout the trajectories as the followers reactively adjust their position to the movements of the leader. Additionally, Figure 8 dynamic starts with Follower A facing the opposite direction of the leader, resulting in higher position error and uncertainty while the rotation error is low. As the Follower rotates, the position uncertainty decreases. During this process, the model generates an erroneous rotation prediction that is accompanied with a spike in the uncertainty prediction. We note that the tracking error for Figure 8 static differs between both agents. This is due to the lack of structure in the lab environment and it affects both robots differently, resulting in better tracking performance for Follower A. The tracking error is consistent for the other trajectories.

Table 7.7 reports quantitative evaluations for the same results, which are in line with the results reported in Table 7.3. Note that the velocities reported are the average velocities of the follower robots. The velocities are consistent for both Figure 8 trajectories, but differ for the rectangle trajectory, as the inner robot moves more slowly than the outer robot.

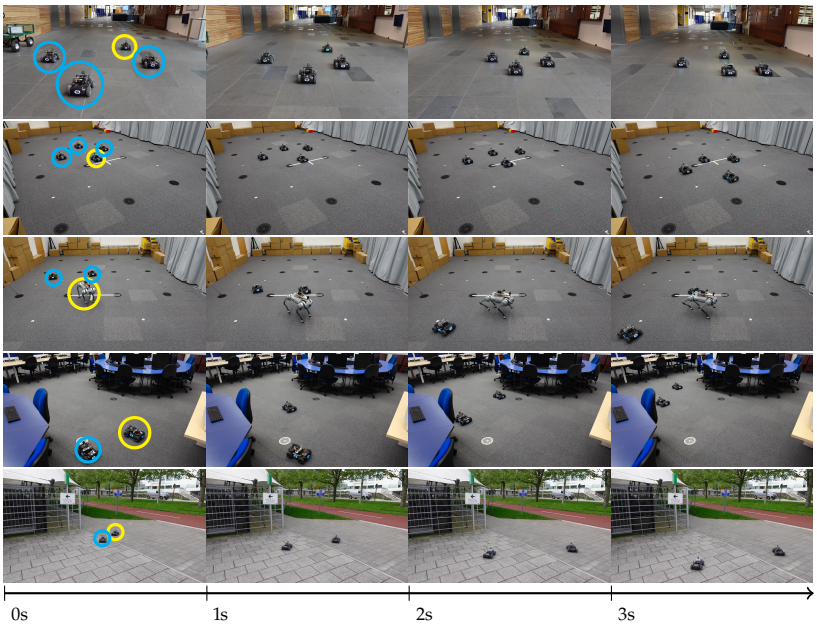


Figure 7.11: We show snapshots of real-world deployments in different scenes with up to four robots. Each row contains six frames from a video recording, each spaced 1 s in time. We indicate the positions of the robots in the first frame, where the leader is circled yellow and the followers blue. The first three samples show indoor scenes, and the bottom sample is an outdoor deployment. The second sample shows a heterogeneous deployment, combining robots of different sizes and dynamics.

7.4 Conclusion

In this chapter, we introduced a first-of-its-kind *cooperative* visual spatial foundation model that leverages existing large vision models and demonstrated its use in multi-robot control. From monocular camera images only, our model is able to output accurate relative pose estimates and BEV representations by aggregating information across neighbouring robots. We validated our model on a custom real-world test set and demonstrated real-time multi-robot control. Our model accurately predicts poses, even without pixel-level correspondences, and correctly predicts BEV representations for obscured regions. In further real-world tests, we demonstrated accurate trajectory tracking, paving the way for more complex vision-only robot tasks. Our model estimates relative poses with a median error of 33 cm and 5.8° for visible edges, and 97 cm and 5.9° for invisible edges, roughly equivalent to one body length of our robot. We have compared our model against two baselines, both of which require visual overlap, and neither of which is able to estimate scale. This positions our solution uniquely amongst related work.

Conclusion

8 Conclusions

8.1 Future work 119
8.2 Closing remarks 120

In this thesis, we pioneer methods for deploying MRS to the real world. We started by introducing a novel multi-agent reinforcement learning scheme that allows training cooperative multi-agent control policies for the CPP problem in a discrete simulation environment. We then train a multi-robot control policy with continuous actions, coordinating five agents through a narrow constriction. We introduce a new multi-robot research platform, the Cambridge RoboMaster, and demonstrate successful decentralised real-world deployment on five robots in the laboratory, using ground-truth state information but exchanging peer-to-peer communications. Next, we explored sensor-guided visual navigation, demonstrating a mobile robot being guided by a network of visual sensor nodes without any ground-truth pose information in a laboratory environment. Lastly, we introduced a novel multi-robot foundation model that allows a decentralised real-world deployment of multiple robots outside the confines of the laboratory with just onboard sensing.

8.1 Future work

The work in this thesis opens several possible frontiers for future research:

- In Chapter 4, we conduct some initial white-box analysis to understand better the communicated learned messages. However, how these insights generalise to other problems and state spaces is unclear. Further research is required to understand better how NNs and GNNs more generally learn and how to deploy robots using them safely to the real world. This is particularly important when it comes to robots working closely together with humans.
- While we investigate the effects of the reality gap on deploying policies trained in simulation to MRS in Chapter 5, we provide no explicit solutions to narrow it. While there was only a slight performance impact on our experiments, it is expected to be more severe in more contested environments or tasks where robots have to move at much higher velocities (e.g. drone flocking). One possible approach is to add domain randomisation to the messaging function of the GNN. Another possibility is to model the behaviour of real-world communication more explicitly in simulators or even

to train on real-world data. As with other robotics problems, the latter is challenging due to a lack of available datasets.

- We show the successful real-world deployment with up to five decentralised robots and up to eight robots in a centralised deployment mode in Chapter 5. Further work on peer-to-peer networking strategies is required to allow scaling vision-based models on robot swarms with significantly more robots. Another solution is to deploy intelligent compression and quantisation strategies during evaluation. Other platforms, such as aerial vehicles, promise more agility and larger swarm sizes.
- State-of-the-art foundation models [145, 252] are typically trained using unsupervised learning. However, we rely on supervised learning to train our model for a class of use cases in MRS, as outlined in Chapter 7. Training the model unsupervised and creating a significantly larger body of widely available real-world data would potentially allow a wider area of applications and improved performance. Furthermore, adding memory capability to the model could make it more capable, significantly expanding its capability past visual overlaps.
- We use transformer-based models in Chapter 7, and while we can deploy them on a capable onboard computer with a GPU in real-time, additional research is necessary to optimise these models further to run on edge devices. Quantisation is one possible approach. Related to this, while simulated data is widely available, equivariant model architectures can make existing real-world more useful and models more sample efficient.

[145]: Oquab et al. (2023), *DINOv2: Learning Robust Visual Features without Supervision*
[252]: Kirillov et al. (2023), *Segment Anything*

8.2 Closing remarks

Deploying MRS to the real world entails various theoretical and systems challenges, requiring robust solutions for and interfaces between perception, control, and multi-robot communication. We have taken significant steps towards addressing these challenges by developing novel multi-robot coordination strategies and cooperative perception modules and showing their efficiency in the field. Doing this gave us new insights into the challenges of deploying MRS to real-world scenarios.

However, we also demonstrated the need for continued research. Expanding scalability is still a topic of discussion, especially systems questions concerning robot-to-robot communication, robustness to real-world uncertainty and partial observability, and advancing the capabilities of multi-robot foundation models. We have set a foundation for future advancements, and we are excited to see robotic systems become increasingly integrated into our everyday environment. It is essential that we continue improving their reliability, safety, and efficiency. With this work,

we present a step towards a vision of fully autonomous, cooperative MRS that can operate in diverse and challenging real-world settings, from warehouses to factories to supermarkets or even unstructured home environments.

Appendix

A

Systems considerations

In this appendix chapter, we consolidate and detail the systems efforts that were essential to the contributions of this thesis. We first provide details on the Cambridge RoboMaster, our custom multi-robot research platform. We continue to show details on the software infrastructure we developed to control and operate a fleet of robots, and then provide details on deploying large NN with millions of parameters to the robots. We also include details on the ad-hoc communication setup.

A.1 Cambridge RoboMaster 125
A.1.1 CAN communication . 125
A.1.2 Compute and components 126
A.2 Fleet management . . . 127
A.2.1 Wireless communication 127
A.2.2 Network 128
A.2.3 Emergency stop 128
A.2.4 On-robot user interface 128
A.2.5 ROS2 drivers 129
A.2.6 Decentralised deployment 129
A.2.7 Multi-robot user interface 129
A.3 Deployment 130
A.3.1 Control stack 130
A.3.2 Camera intrinsics 130
A.3.3 Camera extrinsics 130
A.3.4 Model deployment . . . 131
A.3.5 Ad-hoc WiFi 132

A.1 Cambridge RoboMaster

The RoboMaster is an educational omnidirectional robot platform from DJI. DJI offers an API for the RoboMaster EP, the educational counterpart to the RoboMaster S1, which enables control via a secondary computer (e.g., a Raspberry Pi) through USB. However, this platform is not listed for sale on the DJI store, is unavailable in many parts of the world (including the UK), and at around USD 1500, is significantly costlier than the S1 model. We, therefore, selected the DJI RoboMaster S1 base for our platform. The API, which DJI has published open-source, is software-blocked from the DJI RoboMaster S1. It served as the foundation for our reverse engineering of the communication protocol between the main computer and the sub-controller. In order to replace the computer of the DJI RoboMaster with a custom compute solution, it was necessary to reverse-engineer the CAN protocol, which we describe here. We also provide a detailed breakdown with a bill of material in this section.

`https://github.com/dji-sdk/RoboMaster-SDK`, accessed 03/19/24

A.1.1 CAN communication

DJI uses CAN for hardware layer transport but overlays it with a simplified serial package-based protocol. This design likely addresses the limitations of CAN's 8-byte data size per packet while facilitating multi-node network connectivity, leveraging CAN's collision and arbitration management capabilities. We outline the protocol's structure in Figure A.1, noting its use of a higher-level transport layer to specify node source and destination IDs, sequence counters for package tracking, various command sets and

IDs, and acknowledgements, enabling subscription to specific package IDs for continuous data stream over the network. We release our C++ implementation of the messaging protocol as stand-alone repository.

`https://github.com/proroklab/robomaster_sdk_can`

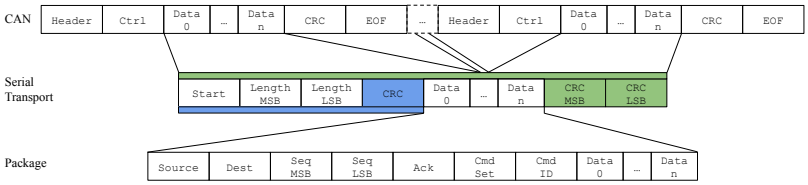


Figure A.1: The CAN-based protocol used in the RoboMasters. DJI uses CAN as the transport layer to allow multiple connected devices on the bus to broadcast information. A serial publisher/subscriber transport protocol is built on top.

A.1.2 Compute and components

We provide a detailed overview of the compute configurations in Figure A.2 and a bill of material in Table A.1.

Table A.1: We provide a cost breakdown for six different computing and sensing configurations for our platform. Optional equipment, such as sensors, are put in parenthesis. All costs are stated in USD.

Item	Raspberry Pi	Orin Nano	Orin NX
Robot Base: RoboMaster S1	549	549	549
Compute	55	255	670
Carrier: AverMedia D131		185	185
WiFi: Intel 9260		15	15
Mem: Micron MTFDKBA256TFK		65	65
Mem: SD Card 256 GB	30		
Infrastructure: DC/DC Converter	20		
Infrastructure: Waveshare CAN Hat	15		
Infrastructure: Acrylics	10	10	10
Infrastructure: Mounting	10	10	10
Infrastructure: OLED Display	(10)	(10)	(10)
WiFi: Netgear A6210	(50)	(50)	(50)
Sensing: Pi HQ Camera	(50)	(50)	(50)
Sensing: Fisheye Lens	(20)	(20)	(20)
Total Base	**689**	1089	1504
Total Optional	(819)	(1219)	(1633)

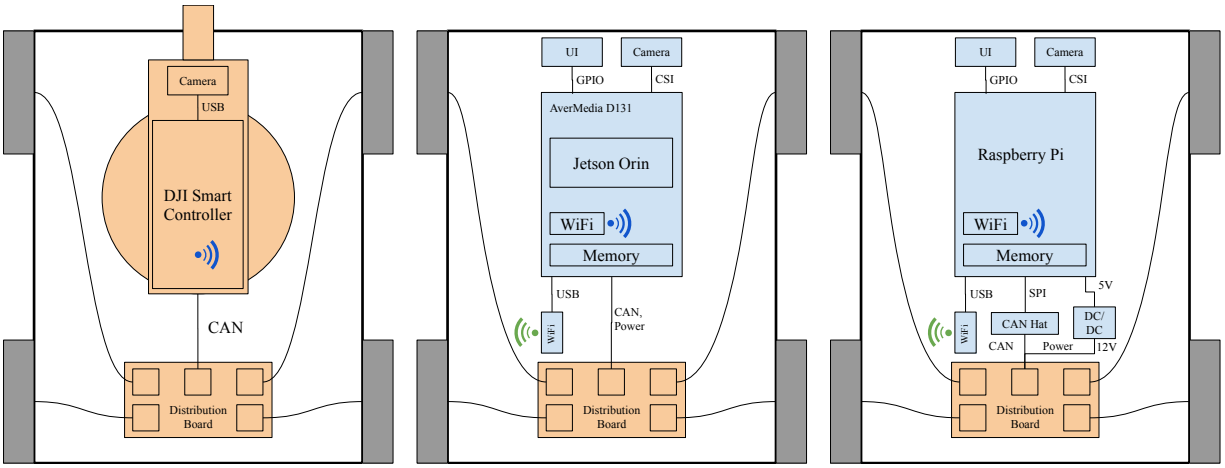


Figure A.2: We modify the RoboMaster S1 Platform by removing the USB Camera, Smart Controller, Blaster and Turret and by replacing them with either a Jetson Orin on an AverMedia D131 carrier board or a Raspberry Pi. The platform is equipped with two WiFi modules, one for infrastructure communication (dark blue) and one specifically chosen for ad-hoc peer-to-peer communication (green).

A.2 Fleet management

Here we describe components of our supporting infrastructure, each of which is a product of over four years of refining and development. The high reliability of these subsystems has been vital to our work in deploying multi-agent research. We provide a high-level overview of the infrastructure in Figure A.3

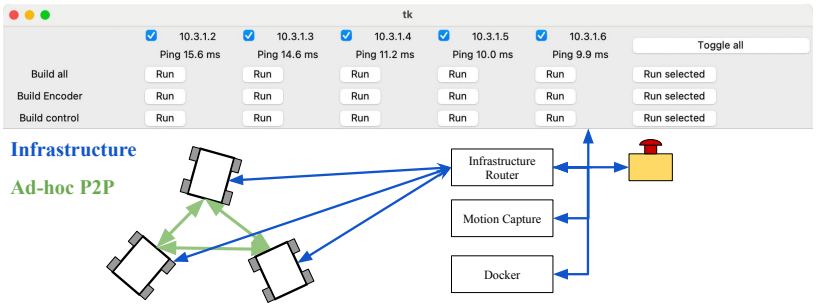


Figure A.3: We utilise two wireless networks, one for infrastructure (blue) and one for ad-hoc peer-to-peer communication between agents. The infrastructure network is used to communicate with fixed infrastructure. The multi-agent user interface to control multiple agents efficiently is displayed at the top. The ad-hoc network is used for experiments requiring decentralised communication.

A.2.1 Wireless communication

Our system employs dual network interfaces to facilitate distinct communication channels: a backbone *infrastructure* network for all participating systems, and, an *ad-hoc* network exclusively for inter-robot communication. For infrastructure-related communication, we utilise an Intel 9260 PCIe module for the Jetson and the integrated WiFi module for the Raspberry Pi that connects to a central WiFi router (described below). This setup integrates with broader infrastructure components, such as desktop systems operating the Multi-Robot Manager UI, a motion capture system, and essential hardware like an emergency stop button. This network also allows for SSH access into the robots, file sharing, and centralised ROS messaging. In scenarios requiring only centralised control, a desktop

machine executes our entire control stack (*Freyja*, described later) for each robot, and transmits low-level wheel commands to each robot via ROS2.

For direct robot-to-robot communication, we use Netgear A6210 USB3.0 modules to establish an ad-hoc multicast network. This configuration does not rely on external infrastructure, enabling seamless peer-to-peer interactions.

A.2.2 Network

The deployment of an enterprise-grade router proved crucial for centralised robot operation. We opted for a Mikrotik Router, which was selected for its superior configurability and flexible antenna setup. We employ a structured IP assignment strategy within the 10.x.y.z address range across all devices and bridge appropriate subnetworks that contain different categories of devices (robots, computers, guest-devices, etc).

Our router and ad-hoc setups typically operate at a broadcast channel capacity of 26 Mb/s, and we typically observe around 1 Mb/s network load when operating five robots concurrently in a centralised mode (transmitting wheel velocity commands to each robot at 50 Hz).

A.2.3 Emergency stop

Supplementary material

We demonstrate this feature here:
Video:

Crash prevention is a priority in high-velocity multi-robot experiments and can be particularly complex to implement in a decentralised framework. Inevitable system latencies in re-establishing SSH sessions and the finite interval between terminating a control process and robot immobilisation necessitate a more reliable stop mechanism. Consequently, we integrate an emergency stop service into our RoboMaster ROS driver, accessible via the infrastructure network, that immediately halts velocity commands at the lowest level. Additionally, we developed a standalone physical emergency stop button, powered by a Raspberry Pi Zero, that connects to the infrastructure network and features colour LEDs to display the robots' connectivity.

A.2.4 On-robot user interface

Incorporating on-robot displays has proven advantageous, especially for operations outside the conventional laboratory setting where an infrastructure network may be absent. These displays, compact and economical at 128×64 pixels, present vital statistics, including CPU and memory usage, CPU temperature, network details (such as IP addresses and interface connectivity), battery status, and the robot's hostname. An

integrated button allows users to navigate through different information pages and, crucially, activates the emergency stop service if the robot is in motion. This feature is particularly useful in field deployments, offering a direct control and monitoring mechanism when network access is unavailable.

A.2.5 ROS2 drivers

We implemented a custom ROS2 Driver that accepts both high-level velocity commands in the robot's body frame as well as low-level wheel speed commands. We use `systemd` to start these drivers automatically during the system boot to make the experimental setup seamless.

A.2.6 Decentralised deployment

For deploying code across multiple robots, we utilise the infrastructure network, employing `rsync` over SSH to synchronise updates from a central workstation to each robot. We extensively leverage Docker containers to remove experimental cross-interference among different users and setups. This approach is particularly beneficial on the NVidia Jetson platform, for which an extensive library of pre-configured containers is available. These containers include environments for ROS, ROS2, PyTorch, and various other learning frameworks, either standalone or in combination. Docker is also configured to manage local image repositories within the infrastructure network, facilitating efficient image sharing among robots.

`https://github.com/dusty-nv/jetson-containers`, accessed on 20/03/2024.

A.2.7 Multi-robot user interface

We developed a configurable Python-based User Interface (UI) operated from a workstation linked to the infrastructure network. This UI facilitates connections to multiple nodes through SSH and supports scripting through a configuration file. It is designed to automate the parallel execution of repetitive commands that normally need to be executed in different terminals on different devices, thus significantly reducing the manual overhead associated with logging into individual robots, sourcing workspaces, and running launch files, especially when identical commands are required on multiple robots. A snapshot of the UI is shown in Figure A.3.

A.3 Deployment

In this section, we provide additional details on the control stack, how we compile and deploy deep models to edge computing infrastructure, and details on the ad-hoc communication setup.

A.3.1 Control stack

A necessary step in successful multi-robot deployments is the dependability of individuals to track their target trajectories accurately. Towards this end, our onboard control stack, *Freyja* [253], comprises of an optimal feedback controller and a library of state filters developed entirely in C++ over the ROS2 middleware. The model-based controller is a linear quadratic regulator (LQR) that regulates the position and velocity of the robot along a trajectory by commanding wheel speeds for the low-level driver. Denoting $\boldsymbol{x} \equiv [\vec{p}, \vec{v}]^\top$ as the state variable (with $\vec{p}$ and $\vec{v}$ as the world-frame position and velocity vectors), we can write a dynamic model for the system as $\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}$, where A and B are system matrices, and $\boldsymbol{u}$ is the applied control input. Then, for some reference state $\boldsymbol{x}_{\text{ref}}$, it is possible to design a matrix K for the control law, $\boldsymbol{u} = -K(\boldsymbol{x} - \boldsymbol{x}_{\text{ref}})$ that stabilises the system to that (possibly time-varying) reference state.

[253]: Shankar et al. (2021), *Freyja: A Full Multirotor System for Agile & Precise Outdoor Flights*

A.3.2 Camera intrinsics

We use cameras with fisheye lenses, resulting in a very large, but distorted, field of view. We use custom-made camera calibration tools based on OpenCV [254, 255] that allow us to automate a majority of the image calibration and cropping procedure required to run the policy. For Chapter 6, we use an equirectangular model and for Chapter 7, a fisheye model. We use a 5×7 checkerboard for both. The fisheye calibration follows the standard procedure of collecting a small image dataset and generating the intrinsics and distortion coefficients. For the equirectangular calibration, we also calibrate the orientation of the camera with respect to the sensor so that camera images are aligned.

[254]: Li et al. (2014), *A Multiple-Camera System Calibration Toolbox Using a Feature Descriptor-Based Calibration Pattern*
[255]: Mei et al. (2007), *Single View Point Omnidirectional Camera Calibration from Planar Grids*

A.3.3 Camera extrinsics

For some of our data visualizations, we use a custom extrinsic camera calibration to be able to project robots tracked by the indoor motion capture system back into the image frame of a third-person view camera. Similar to the intrinsics calibration, we use OpenCV [254, 255] with the Perspective-n-point algorithm to associate points in the camera image frame to points in the world, and thus compute the camera-to-world

transformation. This way, we can visualize arbitrary geometrics and overlay ground truth and estimated data points.

A.3.4 Model deployment

In this section, we provide details on the model trained in Chapter 7.

As outlined in Section 3.4, large models have memory and compute requirements that do not typically align with compute resources available on mobile robots. We chose the smallest distilled version of DinoV2 based on ViT-S/14 with 22.1 M parameters and added additional layers in our model so that the total size is 40 M parameters.

After training in PyTorch, we compile the models for decentralised deployment using TensorRT [256]. One challenge here is that the existing toolchain does not allow for the joint compilation of the model for decentralised deployment, since it considers the whole computation graph in its centralised formulation for fusing. This prohibits us from a joint quantisation [157] for an even faster inference and smaller memory footprint, which would also benefit the communicated embedding. We therefore separately compile the encoder and the downstream model into smaller float representations.

[256]: NVIDIA (2024), *TensorRT*

[157]: Jacob et al. (2018), *Quantization and training of neural networks for efficient integer-arithmetic-only inference*

Using this process, we achieve sub-30 ms inference times. We benchmark the runtime performance of the model on the Jetson Orin NX in Table A.2 by averaging 100 sequential forward passes. We compare 16-bit and 32-bit floats evaluated on the GPU or CPU and compilation with TensorRT, a proprietary NVidia tool for the ahead-of-time compilation of neural networks. The encoder model, generating the communicated embedding, has a total of 30 M parameters and takes ca. 20 ms for one forward pass as FP16 optimised with Torch TensorRT on the GPU. This is 50× faster than running the same model on the CPU. We also note that changing from 32-bit floats to 16-bit floats results in a 2.75× speedup. The pose estimation model, processing a pair of embeddings into a relative pose, has a total of 6 M parameters and takes ca. 4 ms for one forward pass. The model is run in a custom C++ environment integrated with ROS2 Humble and an ad-hoc WiFi network for inter-node communication to communicate image embeddings, achieving a 15 Hz processing rate for the whole pipeline, including image pre-processing, model evaluation, communication and pose prediction.

Type	Mode	Dev.	f_{enc}		f_{pose}	
FP16	TRT	GPU	**19.22 ms**	±0.92 ms	**4.15 ms**	±0.49 ms
FP16	JIT	GPU	36.79 ms	±1.33 ms	8.04 ms	±0.73 ms
FP32	JIT	GPU	99.32 ms	±1.56 ms	13.57 ms	±1.10 ms
FP32	JIT	CPU	952.04 ms	±116.72 ms	248.41 ms	±45.98 ms

Table A.2: Model runtime evaluation for $S = 128$ and $F = 24$.

A.3.5 Ad-hoc WiFi

Image embeddings are sent between nodes at a rate of 15hz, with each being just over 6KiB. The fully distributed nature of our system matches well with a broadcast communications topology, which theoretically can permit operation at scale. Broadcast messaging has a significant drawback, however, in that modern wireless data networking standards (IEEE 802.11/WiFi) fall back to very low bit rates to ensure maximum reception probability.

802.11 uses low bit rates for broadcast due to a lack of acknowledgement messages, which are the typical feedback mechanism for data rate control and retransmissions. The probability of a frame being lost due to the underlying CSMA/CA frame scheduler is non-trivial (3%+); a condition which worsens with increasing participating network nodes.

A higher level system, such as ROS2 communications middleware package CycloneDDS can manage retransmissions, however currently available systems are not well suited to the requirements of a decentralised robotic network. In particular, they lack the ability to detect network overload conditions, which can result in retransmissions that further load the network, finally resulting in a rapid increase in packet loss rates, which renders the network as a whole unusable. While it is possible to tune such communications packages manually, the best parameter values are sensitive to highly variable deployment configuration parameters, including ambient wireless network traffic. Manual tuning of network hardware, such as forcing increased broadcast bit rates, can alleviate this problem but is generally not as resilient as scale.

Our custom messaging protocol is targeted to this specific use case and includes the ability to dynamically back off messaging load based upon detected network conditions. Our system also uses a shared slot TDMA message scheduling system, which works with the 802.11 frame scheduler to maximise frame reception probability without compromising overall data rates.

B

Visual navigation

We provide additional qualitative results for simulation and include all real-world episodes.

B.1 Simulation **133**
B.2 Real-world **136**

B.1 Simulation

We report a broader selection of path evaluations in simulation, specifically NLOS, or non-line-of-sight, in Figure B.1. We show samples from generalization to larger environments in Figure B.2.

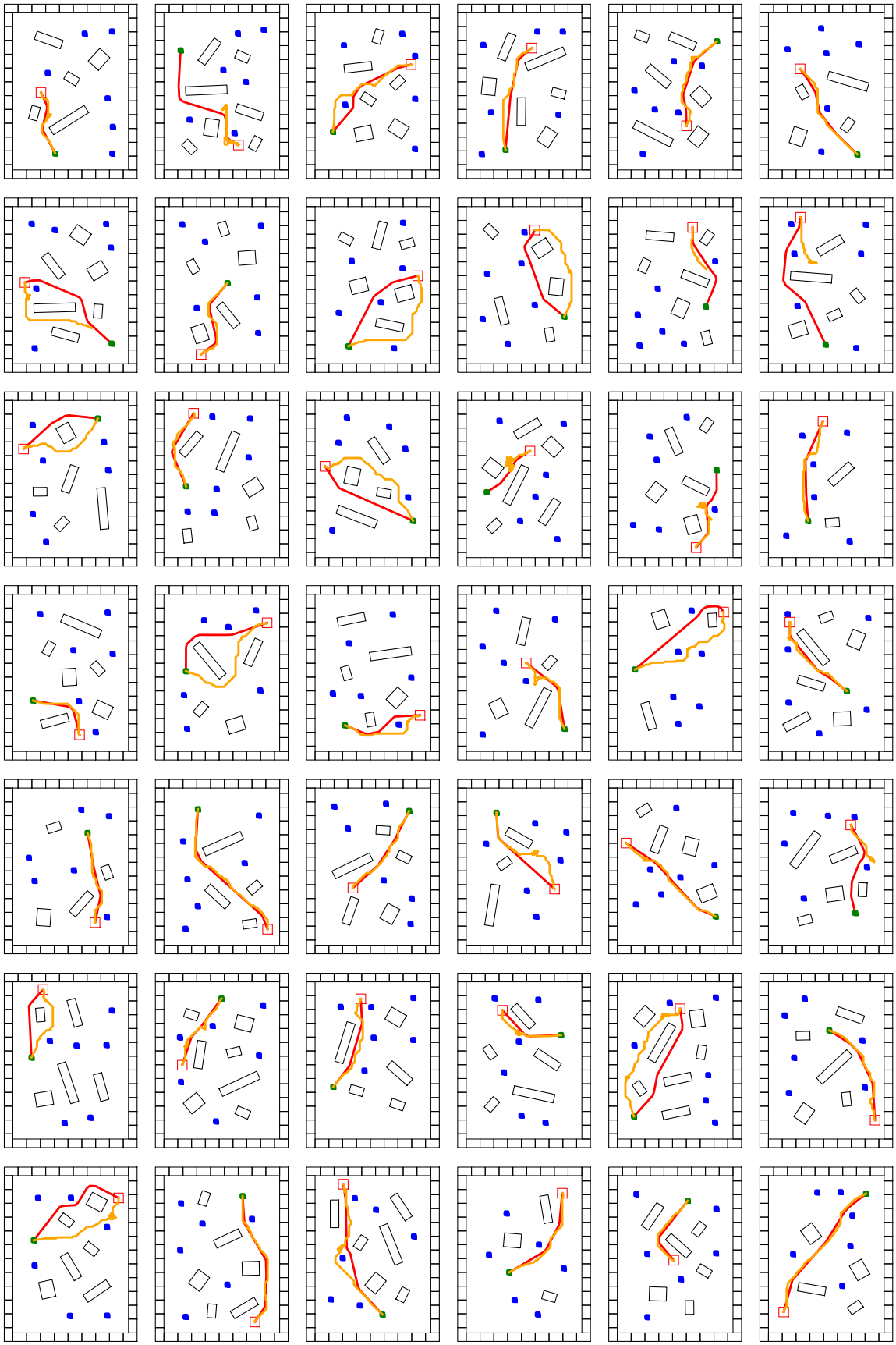


Figure B.1: A selection of policy evaluations for NLOS configurations in simulation. Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π.

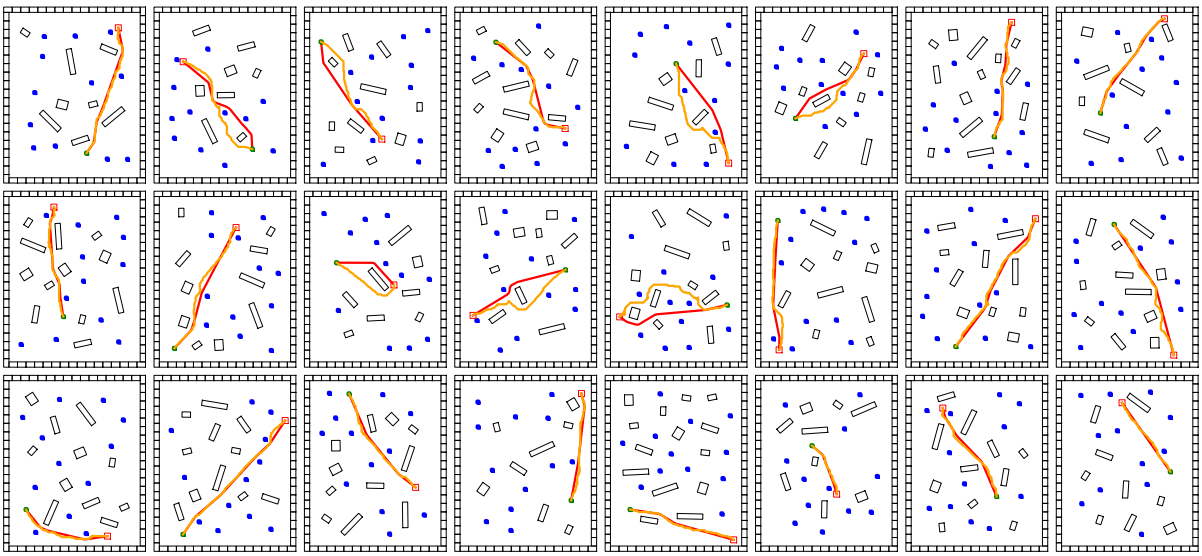


Figure B.2: Generalizability to larger environments and larger number of agents $N = 13$ for GNN layers $L = 2$ and communication range $D_S = 3.5$ m (SPL 0.91). Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π.

B.2 Real-world

We show all evaluation runs from the three real-world environments in Figure B.3 (Environment A), Figure B.4 (Environment B), and Figure B.5 (Environment C).

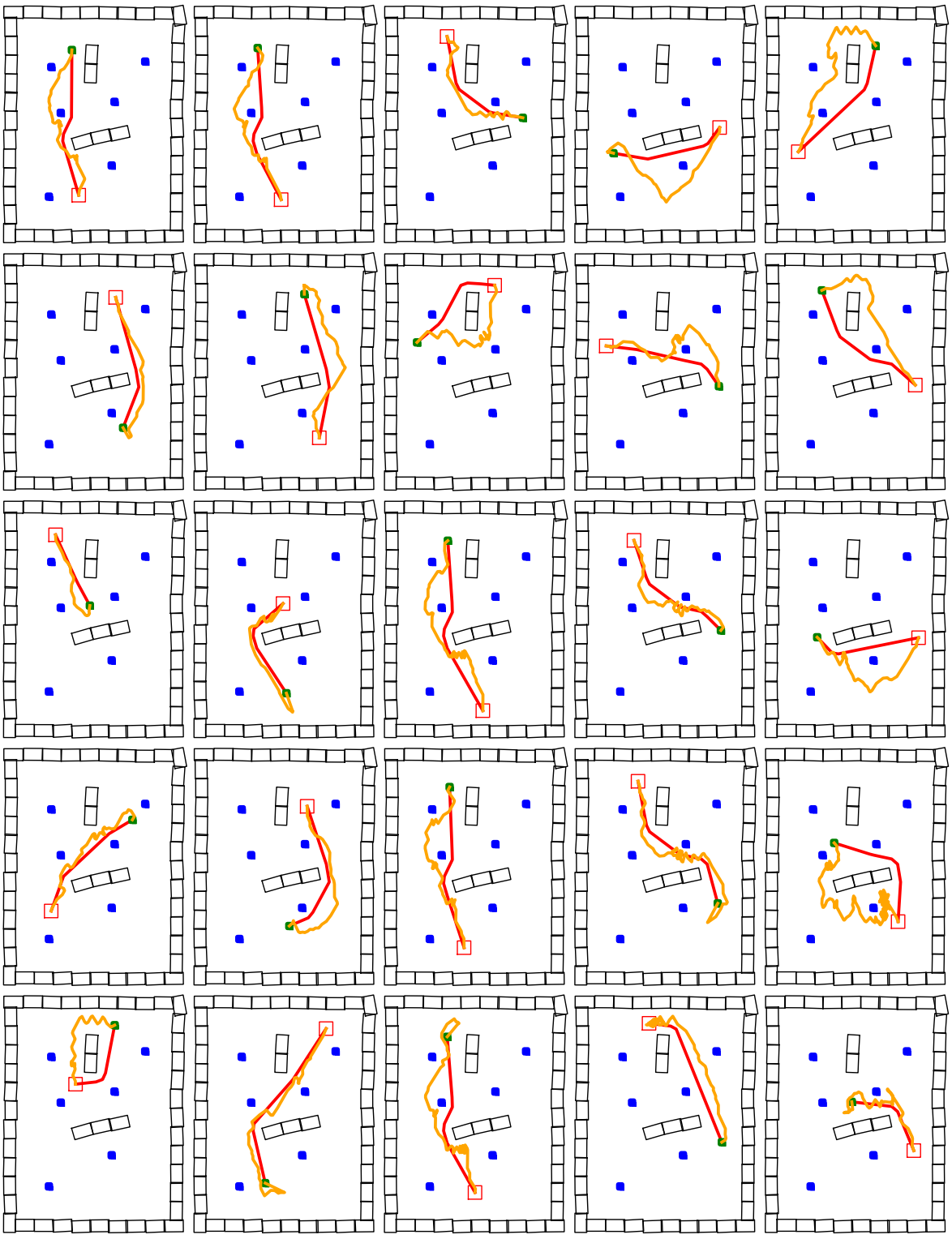


Figure B.3: All real-world evaluations for Environment A. Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π.

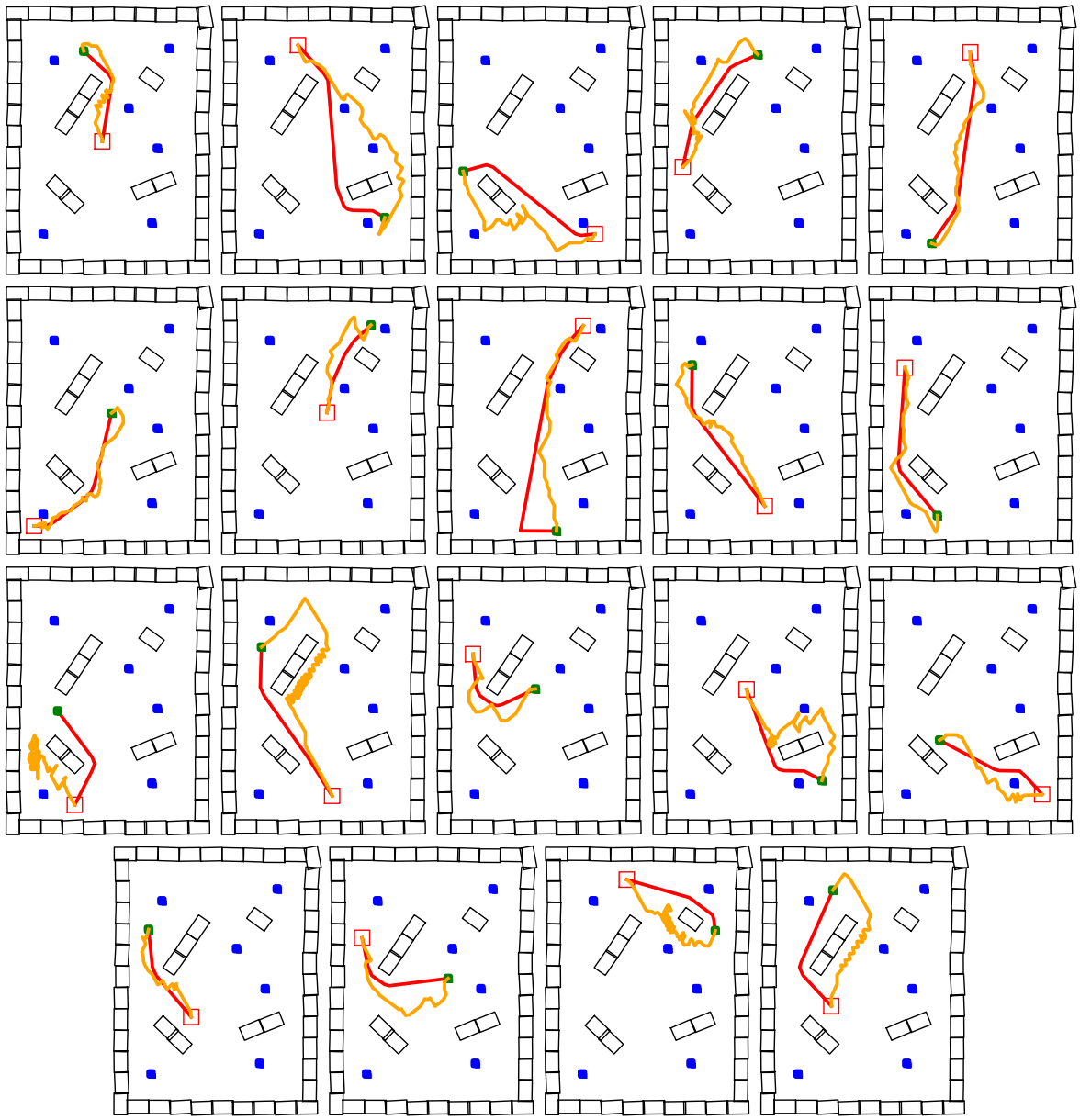


Figure B.4: All real-world evaluations for Environment B. Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π.

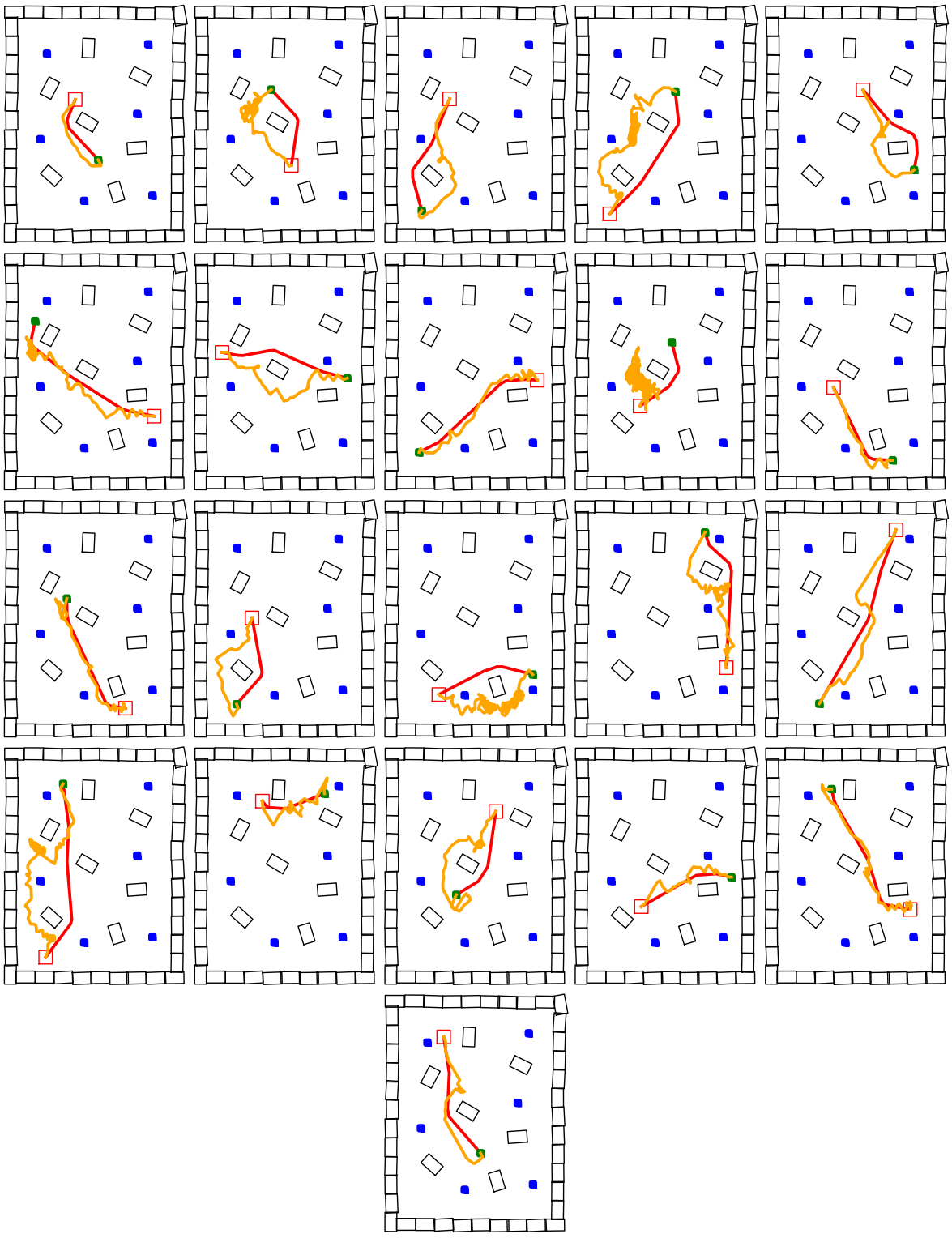


Figure B.5: All real-world evaluations for Environment C. Blue squares indicate sensor positions, the green square the target position q^G, the red path the robot's initial position q_0^R, the red path the shortest path computed by the expert and the orange path the path chosen by the policy π.

C

Visual spatial foundation models

C.1 Simulation

C.1 Simulation 141
C.2 Real-world 144
C.3 Real-world pose control . 145

We visualize four additional qualitative samples from the simulation dataset $\mathcal{D}_{\text{Test}}^{\text{Sim}}$ in Figure C.1, Figure C.2, Figure C.3, Figure C.4, and Figure C.5.

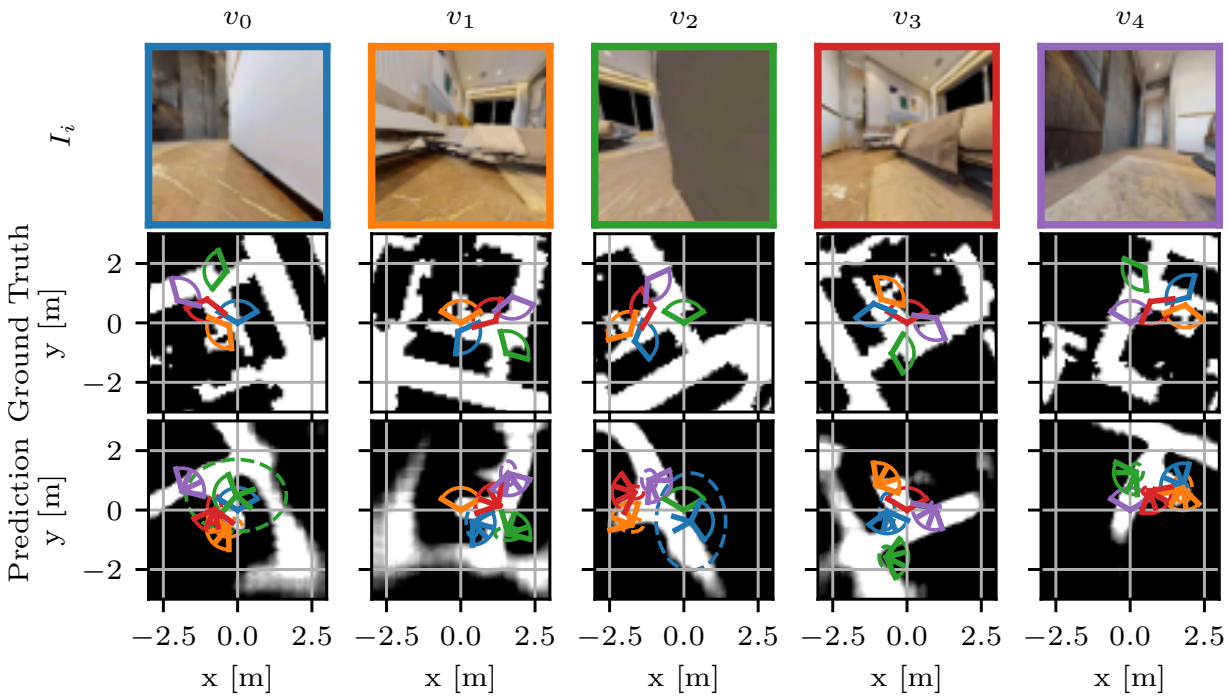


Figure C.1: Sample A from $\mathcal{D}_{\text{Test}}^{\text{Sim}}$.

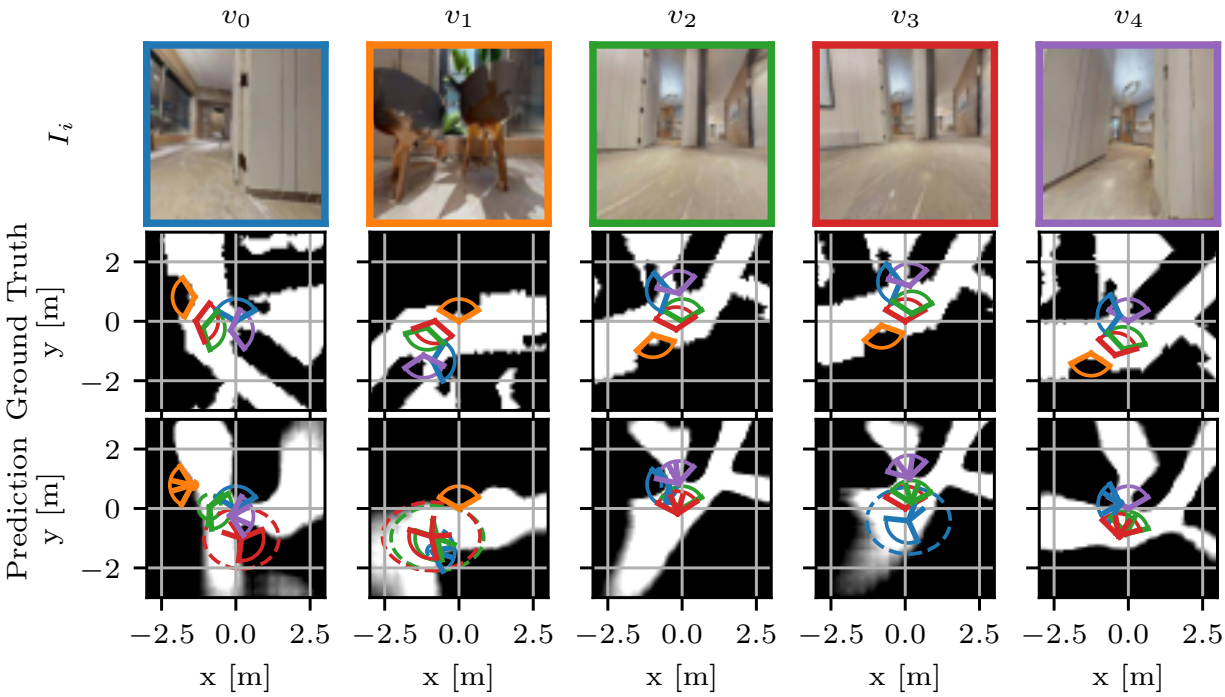


Figure C.2: Sample B from $\mathcal{D}_{\text{Test}}^{\text{Sim}}$.

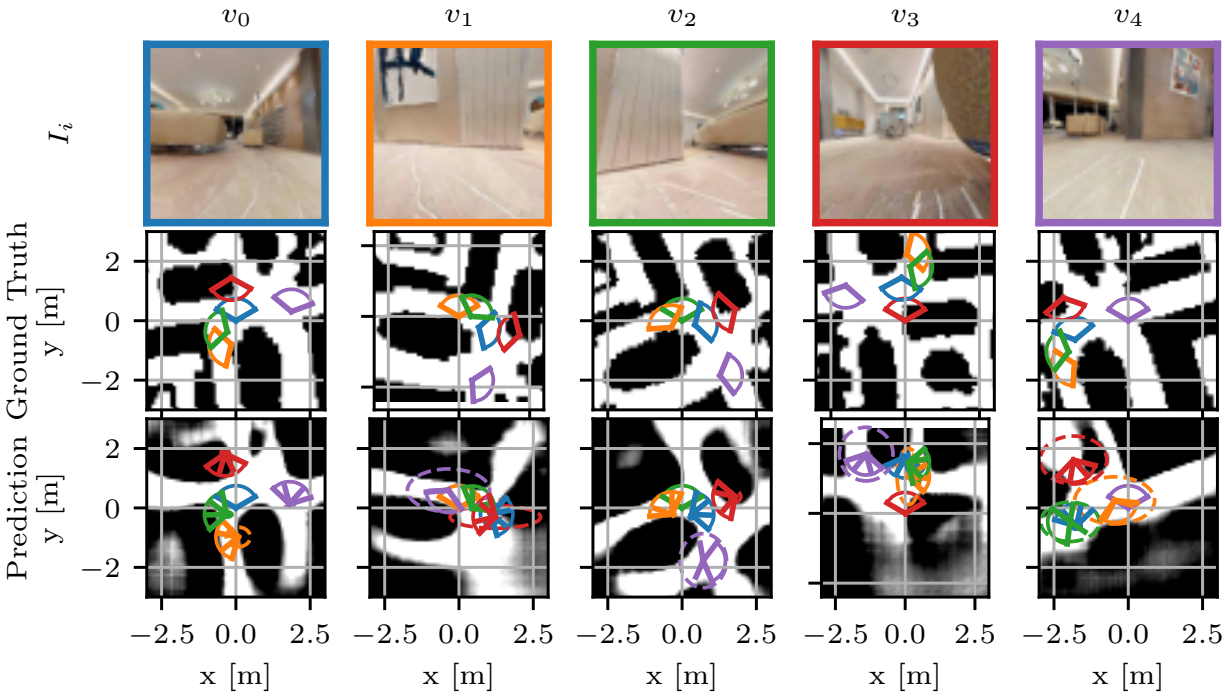


Figure C.3: Sample C from $\mathcal{D}_{\text{Test}}^{\text{Sim}}$.

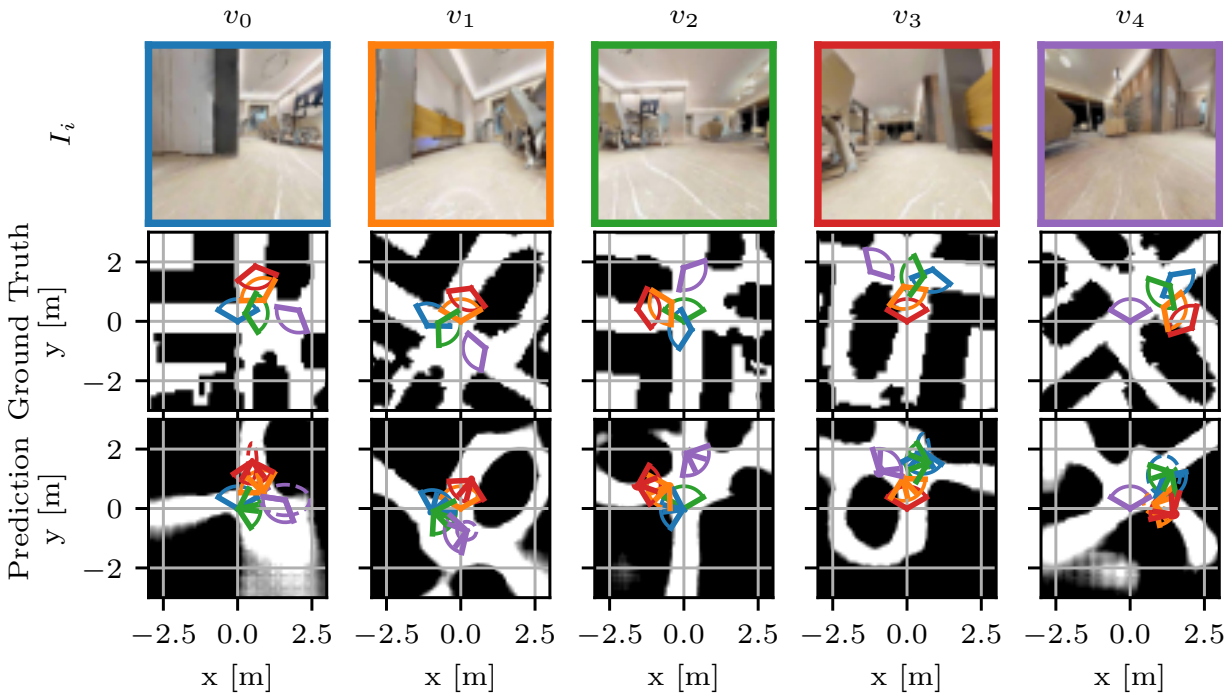


Figure C.4: Sample D from $\mathcal{D}_{\text{Test}}^{\text{Sim}}$.

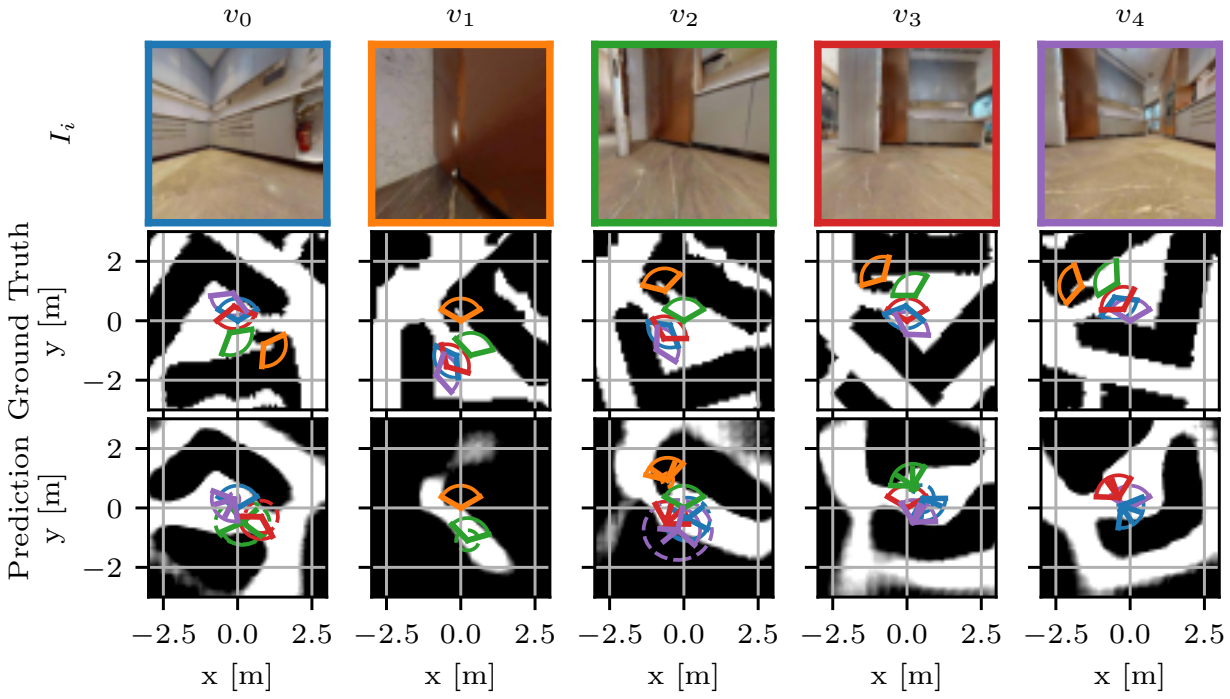


Figure C.5: Sample E from $\mathcal{D}_{\text{Test}}^{\text{Sim}}$.

C.2 Real-world

We visualize four additional qualitative samples from the real-world dataset $\mathcal{D}_{\text{Test}}^{\text{Real}}$ in Figure C.6 and Figure C.7.

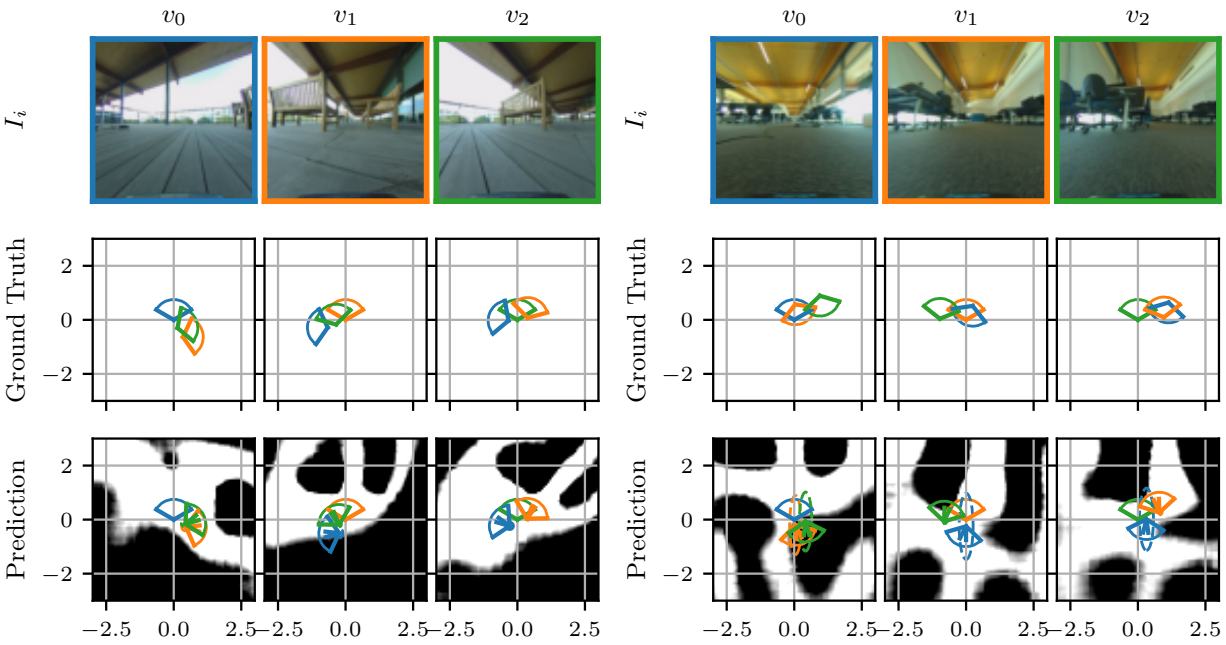


Figure C.6: We visualize a sample from our custom real-world testset $\mathcal{D}_{\text{Test}}^{\text{Real}}$ for three nodes, similar to the simulation sample in Figure 7.4. We do not show the ground-truth BEV segmentation since we do not have labels for this. Additionally to the pose predictions $\hat{\mathbf{p}}^{i,ij}$, $\hat{\mathbf{R}}^{i,ij}$, we also visualize the predicted uncertainties $\sigma_{\text{p}}^{2,i}ij$ and $\sigma_{\text{R}}^{2,i}ij$.

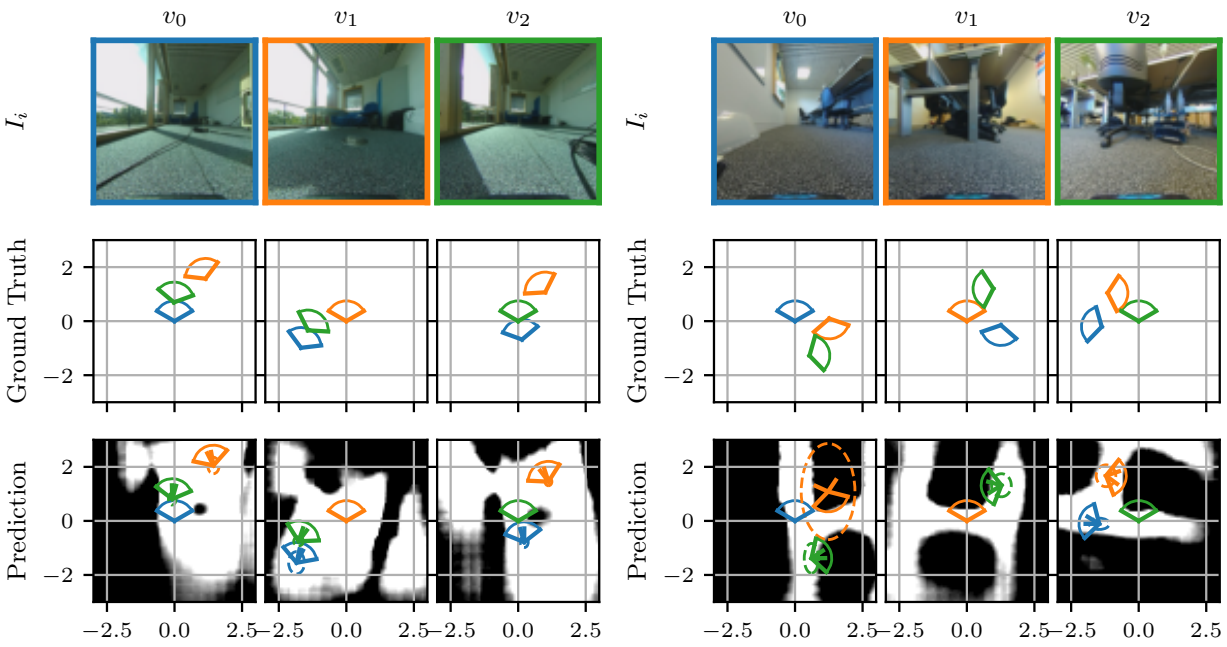


Figure C.7: Sample C and D from $\mathcal{D}_{\text{Test}}^{\text{Real}}$.

C.3 Real-world pose control

We provide an extended overview of real-world deployment snapshots in Figure C.8.

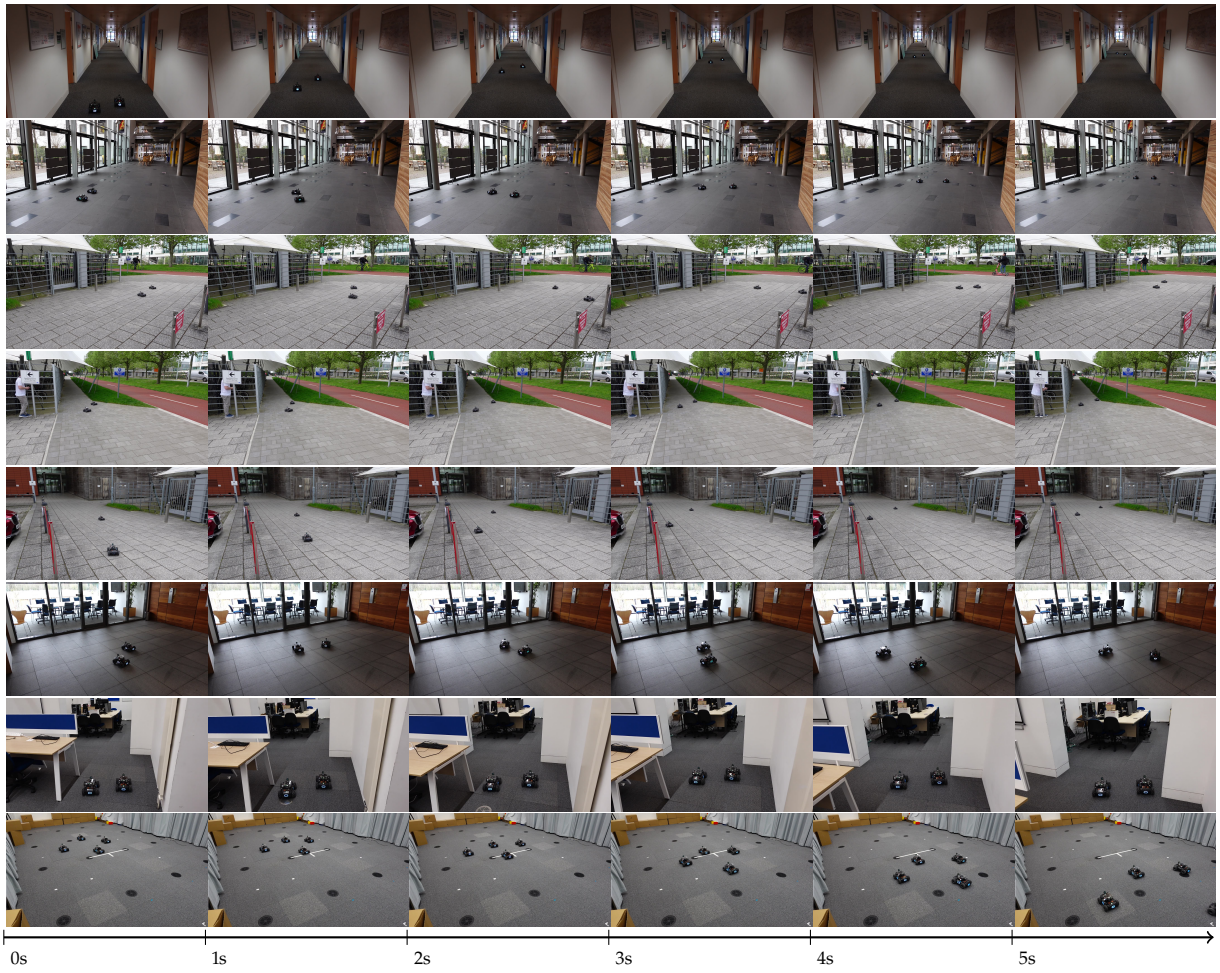


Figure C.8: We show additional snapshots of real-world deployments in different scenes with up to four robots. Each row contains six frames from a video recording, each spaced 1 s in time.

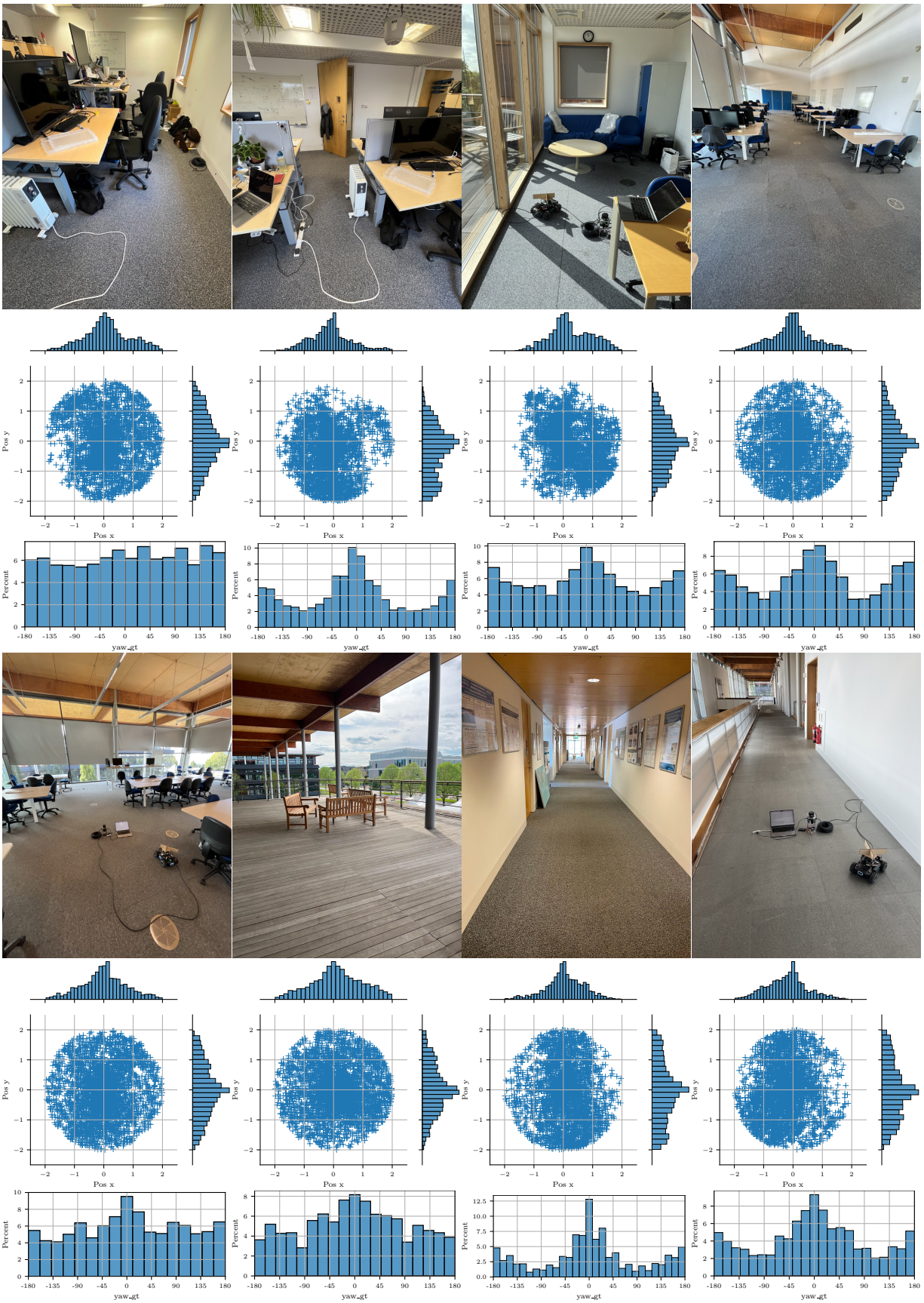


Figure C.9: We show additional statistics for the real-world dataset $\mathcal{D}_{\text{Test}}^{\text{Real}}$. In particular, we show the distribution of positions for each scene and the distribution of relative angles between nodes.

Bibliography

[1] Peter Stone and Manuela Veloso. 'Multiagent Systems: A Survey from a Machine Learning Perspective'. In: *Autonomous Robots* 8.3 (2000), pp. 345–383. DOI: 10.1023/A:1008942012299. (Visited on 11/06/2024) (cit. on pp. iii, 11–14, 27).

[2] 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 (cit. on p. iii).

[3] Qingbiao Li, Fernando Gama, Alejandro Ribeiro and Amanda Prorok. 'Graph neural networks for decentralized multi-robot path planning'. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 11785–11792 (cit. on pp. iii, 4, 33, 41, 61, 86).

[4] Fernando Gama, Ekaterina Tolstaya and Alejandro Ribeiro. 'Graph Neural Networks for Decentralized Controllers'. In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2021, pp. 5260–5264. DOI: 10.1109/ICASSP39728.2021.9414563 (cit. on pp. iii, 4, 33).

[5] Nick Jakobi, Phil Husbands and Inman Harvey. 'Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics'. In: *Advances in Artificial Life*. Vol. 929. Springer, 1995 (cit. on pp. iii, 4, 29).

[6] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba and Pieter Abbeel. 'Domain randomization for transferring deep neural networks from simulation to the real world'. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2017, pp. 23–30 (cit. on pp. iii, 37).

[7] Jan Blumenkamp and Amanda Prorok. 'The Emergence of Adversarial Communication in Multi-Agent Reinforcement Learning'. In: *Proceedings of the 2020 Conference on Robot Learning*. Ed. by Jens Kober, Fabio Ramos and Claire Tomlin. Vol. 155. Proceedings of Machine Learning Research. PMLR, Nov. 2021, pp. 1394–1414 (cit. on pp. iii, 4, 7, 8, 42).

[8] Jan Blumenkamp, Ajay Shankar, Matteo Bettini, Joshua Bird and Amanda Prorok. *The Cambridge RoboMaster: An Agile Multi-Robot Research Platform*. 2024 (cit. on pp. iii, 7, 8, 108).

[9] Jan Blumenkamp, Steven Morad, Jennifer Gielis, Qingbiao Li and Amanda Prorok. 'A Framework for Real-World Multi-Robot Systems Running Decentralized GNN-Based Policies'. In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 8772–8778. DOI: 10.1109/ICRA46639.2022.9811744 (cit. on pp. iii, 7, 9).

[10] Jan Blumenkamp, Qingbiao Li, Binyu Wang, Zhe Liu and Amanda Prorok. 'See What the Robot Can't See: Learning Cooperative Perception for Visual Navigation'. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 7333–7340 (cit. on pp. iii, 8, 9).

[11] Jan Blumenkamp, Steven Morad, Jennifer Gielis and Amanda Prorok. *CoViS-Net: A Cooperative Visual Spatial Foundation Model for Multi-Robot Applications*. 2024 (cit. on pp. iii, 8, 9).

[12] Benjamin Hudson, Qingbiao Li, Matthew Malencia and Amanda Prorok. 'Graph Neural Network Guided Local Search for the Traveling Salesperson Problem'. In: *International Conference on Learning Representations (ICLR)* (2022) (cit. on p. 3).

[13] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V Snyder and Martin Takáč. 'Reinforcement learning for solving the vehicle routing problem'. In: *arXiv preprint arXiv:1802.04240* (2018) (cit. on p. 3).

[14] Jingjin Yu and Steven M LaValle. 'Structure and intractability of optimal multi-robot path planning on graphs'. In: *AAAI*. 2013 (cit. on p. 3).

[15] Vishnu R Desaraju and Jonathan P How. 'Decentralized path planning for multi-agent teams with complex constraints'. In: *Autonomous Robots* 32.4 (2012), pp. 385–403 (cit. on p. 3).

[16] 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*. 2008, pp. 1928–1935 (cit. on pp. 4, 32).

[17] Emanuel Todorov, Tom Erez and Yuval Tassa. 'MuJoCo: A physics engine for model-based control'. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vilamoura-Algarve, Portugal: IEEE, Oct. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109. (Visited on 25/08/2024) (cit. on pp. 4, 28).

[18] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. 2016. URL: http://pybullet.org (cit. on pp. 4, 28).

[19] Manolis Savva et al. 'Habitat: A Platform for Embodied AI Research'. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019 (cit. on pp. 4, 28, 102).

[20] J. T. Springenberg, A. Dosovitskiy, T. Brox and M. Riedmiller. 'Striving for Simplicity: The All Convolutional Net'. In: *ICLR (workshop track)*. 2015 (cit. on pp. 7, 54).

[21] Jan Blumenkamp. *Learning interpretable explicit communication for the multi-agent coverage problem*. Tech. rep. University of Cambridge, July 2020, p. 70 (cit. on pp. 7, 9, 42).

[22] Shiva Shahrokhi, Lillian Lin, Chris Ertel, Mable Wan and Aaron T. Becker. 'Steering a Swarm of Particles Using Global Inputs and Swarm Statistics'. In: *IEEE Transactions on Robotics* 34.1 (Feb. 2018), pp. 207–219. DOI: 10.1109/TRO.2017.2769094. (Visited on 31/01/2025) (cit. on p. 13).

[23] Shounak Roychowdhury, Neeraj Arora and Sandip Sen. 'Effects of local information on group behavior'. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*. AAAI'96. AAAI Press, 1996, p. 1406 (cit. on p. 13).

[24] Owen W. Holland. 'Multiagent systems: Lessons from social insects and collective robotics'. In: *Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium* (Mar. 1996), pp. 57–62 (cit. on p. 13).

[25] Claudia V. Goldman and Jeffrey S. Rosenschein. 'Emergent coordination through the use of cooperative state-changing rules'. In: *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence*. AAAI'94. AAAI Press, 1994, pp. 408–413 (cit. on p. 13).

[26] Tucker Balch and Ronald C. Arkin. 'Communication in reactive multiagent robotic systems'. In: *Autonomous Robots* 1.1 (Mar. 1994), pp. 27–52. DOI: 10.1007/BF00735341 (cit. on p. 14).

[27] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016 (cit. on p. 16).

[28] Vinod Nair and Geoffrey E Hinton. 'Rectified linear units improve restricted boltzmann machines'. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814 (cit. on pp. 16, 54).

[29] Kurt Hornik. 'Approximation capabilities of multilayer feedforward networks'. en. In: *Neural Networks* 4.2 (1991), pp. 251–257. DOI: 10.1016/0893-6080(91)90009-T. (Visited on 28/08/2024) (cit. on p. 16).

[30] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams. 'Learning representations by back-propagating errors'. en. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0. (Visited on 28/08/2024) (cit. on p. 16).

[31] J. Kiefer and J. Wolfowitz. 'Stochastic Estimation of the Maximum of a Regression Function'. en. In: *The Annals of Mathematical Statistics* 23.3 (Sept. 1952), pp. 462–466. DOI: 10.1214/aoms/1177729392. (Visited on 28/08/2024) (cit. on p. 16).

[32] Michael M Bronstein, Joan Bruna, Taco Cohen and Petar Veličković. 'Geometric deep learning: Grids, groups, graphs, geodesics, and gauges'. In: *arXiv preprint arXiv:2104.13478* (2021) (cit. on pp. 17–21).

[33] Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann et al. 'A deep learning approach to antibiotic discovery'. In: *Cell* 180.4 (2020), pp. 688–702 (cit. on p. 19).

[34] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire et al. 'Eta prediction with graph neural networks in google maps'. In: *Proceedings of the 30th ACM international conference on information & knowledge management*. 2021, pp. 3767–3776 (cit. on p. 19).

[35] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton and Jure Leskovec. 'Graph convolutional neural networks for web-scale recommender systems'. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 974–983 (cit. on p. 19).

[36] Junheng Hao, Tong Zhao, Jin Li, Xin Luna Dong, Christos Faloutsos, Yizhou Sun and Wei Wang. 'P-companion: A principled framework for diversified complementary product recommendation'. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 2517–2524 (cit. on p. 19).

[37] Michaël Defferrard, Xavier Bresson and Pierre Vandergheynst. 'Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering'. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. USA: Curran Associates Inc., 2016, pp. 3844–3852 (cit. on p. 20).

[38] Thomas N. Kipf and Max Welling. 'Semi-Supervised Classification with Graph Convolutional Networks'. In: *International Conference on Learning Representations (ICLR)*. 2017 (cit. on p. 20).

[39] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu and Kilian Weinberger. 'Simplifying graph convolutional networks'. In: *International conference on machine learning*. PMLR, 2019, pp. 6861–6871 (cit. on p. 20).

[40] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò and Yoshua Bengio. 'Graph Attention Networks'. In: *International Conference on Learning Representations* (2018) (cit. on p. 20).

[41] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende and Koray kavukcuoglu. 'Interaction Networks for Learning about Objects, Relations and Physics'. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, pp. 4509–4517 (cit. on p. 20).

[42] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals and George E Dahl. 'Neural Message Passing for Quantum Chemistry'. In: *ICML*. 2017 (cit. on pp. 20, 71).

[43] Peter Battaglia et al. 'Relational inductive biases, deep learning, and graph networks'. In: *arXiv* (2018) (cit. on p. 20).

[44] Petar Veličković. 'Everything is connected: Graph neural networks'. In: *Current Opinion in Structural Biology* 79 (2023), p. 102538 (cit. on p. 20).

[45] Ronald A. Howard. *Dynamic programming and Markov processes*. eng. 6. print. Cambridge, Mass: M.I.T. Pr, 1970 (cit. on p. 21).

[46] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. Cambridge, Mass: MIT Press, 1998 (cit. on pp. 22–25).

[47] K.J Åström. 'Optimal control of Markov processes with incomplete state information'. en. In: *Journal of Mathematical Analysis and Applications* 10.1 (Feb. 1965), pp. 174–205. DOI: 10.1016/0022-247X(65)90154-X. (Visited on 23/08/2024) (cit. on pp. 22, 23).

[48] L. S. Shapley. 'Stochastic Games'. en. In: *Proceedings of the National Academy of Sciences* 39.10 (Oct. 1953), pp. 1095–1100. DOI: 10.1073/pnas.39.10.1095. (Visited on 24/08/2024) (cit. on p. 22).

[49] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. 1st ed. 2016. SpringerBriefs in Intelligent Systems, Artificial Intelligence, Multiagent Systems, and Cognitive Robotics. Cham: Springer International Publishing : Imprint: Springer, 2016 (cit. on p. 23).

[50] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin A. Riedmiller. 'Playing Atari with Deep Reinforcement Learning'. In: *CoRR* abs/1312.5602 (2013) (cit. on p. 25).

[51] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver and Koray Kavukcuoglu. 'Asynchronous Methods for Deep Reinforcement Learning'. In: *CoRR* abs/1602.01783 (2016) (cit. on p. 26).

[52] Vijay R. Konda and John N. Tsitsiklis. 'Actor-Critic Algorithms'. In: *Advances in Neural Information Processing Systems 12*. Ed. by S. A. Solla, T. K. Leen and K. Müller. MIT Press, 2000, pp. 1008–1014 (cit. on pp. 26, 46).

[53] Jan Peters and Stefan Schaal. 'Natural Actor-Critic'. In: *Neurocomputing* 71.7 (2008), pp. 1180–1190. DOI: https://doi.org/10.1016/j.neucom.2007.11.026 (cit. on p. 26).

[54] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan and Philipp Moritz. 'Trust Region Policy Optimization'. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1889–1897 (cit. on p. 26).

[55] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford and Oleg Klimov. 'Proximal Policy Optimization Algorithms'. In: *CoRR* abs/1707.06347 (2017) (cit. on pp. 26, 27, 46).

[56] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel and Igor Mordatch. 'Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments'. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 6382–6393 (cit. on pp. 27, 31).

[57] Haonan Yu. *On Sim2Real Transfer in Robotics*. 2024. URL: https://www.haonanyu.blog/post/sim2real/ (cit. on p. 28).

[58] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun and Davide Scaramuzza. 'Learning high-speed flight in the wild'. en. In: *Science Robotics* 6.59 (Oct. 2021), eabg5810. DOI: 10.1126/scirobotics.abg5810. (Visited on 27/09/2024) (cit. on p. 28).

[59] Laetitia Matignon, Guillaume J. Laurent and Nadine Le Fort-Piat. 'Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems'. In: *The Knowledge Engineering Review* 27.1 (2012), pp. 1–31. DOI: 10.1017/S0269888912000057 (cit. on p. 31).

[60] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster and Shimon Whiteson. 'QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning'. In: *ICML*. 2018, pp. 4292–4301 (cit. on p. 31).

[61] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli and Shimon Whiteson. 'Counterfactual Multi-Agent Policy Gradients'. In: *Thirty-second AAAI conference on artificial intelligence*. 2018 (cit. on pp. 31, 43).

[62] Jayesh K. Gupta, Maxim Egorov and Mykel Kochenderfer. 'Cooperative Multi-agent Control Using Deep Reinforcement Learning'. In: *Autonomous Agents and Multiagent Systems*. Ed. by Gita Sukthankar and Juan A. Rodriguez-Aguilar. Cham: Springer International Publishing, 2017, pp. 66–83 (cit. on p. 31).

[63] Landon Kraemer and Bikramjit Banerjee. 'Multi-agent reinforcement learning as a rehearsal for decentralized planning'. In: *Neurocomputing* 190 (2016), pp. 82–94. DOI: https://doi.org/10.1016/j.neucom.2016.01.031 (cit. on p. 31).

[64] 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 29*. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon and R. Garnett. Curran Associates, Inc., 2016, pp. 2137–2145 (cit. on pp. 31, 41).

[65] Igor Mordatch and Pieter Abbeel. 'Emergence of grounded compositional language in multi-agent populations'. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018 (cit. on p. 31).

[66] Darío Maravall, Javier de Lope and Raúl Domínguez. 'Coordination of Communication in Robot Teams by Reinforcement Learning'. In: *Foundations on Natural and Artificial Computation*. Ed. by José Manuel Ferrández, José Ramón Álvarez Sánchez, Félix de la Paz and F. Javier Toledo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 156–164 (cit. on p. 32).

[67] Chongjie Zhang and Victor Lesser. 'Coordinating Multi-Agent Reinforcement Learning with Limited Communication'. In: *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*. AAMAS '13. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 1101–1108 (cit. on p. 32).

[68] Craig W. Reynolds. 'Flocks, Herds and Schools: A Distributed Behavioral Model'. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. New York, NY, USA: Association for Computing Machinery, 1987, pp. 25–34. DOI: 10.1145/37401.37406 (cit. on pp. 32, 97).

[69] Kwang-Kyo Oh, Myoung-Chul Park and Hyo-Sung Ahn. 'A survey of multi-agent formation control'. In: *Automatica* 53 (2015), pp. 424–440 (cit. on p. 32).

[70] Guni Sharon, Roni Stern, Ariel Felner and Nathan R Sturtevant. 'Conflict-based search for optimal multi-agent pathfinding'. In: *Artificial intelligence* 219 (2015), pp. 40–66 (cit. on pp. 32, 97).

[71] Alcherio Martinoli, Kjerstin Easton and William Agassounon. 'Modeling swarm robotic systems: A case study in collaborative distributed manipulation'. In: *The International Journal of Robotics Research* 23.4-5 (2004), pp. 415–436 (cit. on p. 32).

[72] Javier Alonso-Mora, Ross Knepper, Roland Siegwart and Daniela Rus. 'Local motion planning for collaborative multi-robot manipulation of deformable objects'. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 5495–5502. DOI: 10.1109/ICRA.2015.7139967 (cit. on p. 32).

[73] Alexander Zelinsky, Ray A Jarvis, JC Byrne, Shinichi Yuta et al. 'Planning paths of complete coverage of an unstructured environment by a mobile robot'. In: *Proceedings of international conference on advanced robotics*. Vol. 13. Citeseer, 1993, pp. 533–538 (cit. on p. 32).

[74] Stephen L. Smith, Mac Schwager and Daniela Rus. 'Persistent Robotic Tasks: Monitoring and Sweeping in Changing Environments'. In: *IEEE Transactions on Robotics* 28.2 (2012), pp. 410–426. DOI: 10.1109/TRO.2011.2174493 (cit. on p. 32).

[75] Xin Zhou et al. 'Swarm of micro flying robots in the wild'. en. In: *Science Robotics* 7.66 (May 2022), eabm5954. DOI: 10.1126/scirobotics.abm5954. (Visited on 10/06/2024) (cit. on p. 32).

[76] Gábor Vásárhelyi, Csaba Virágh, Gergő Somorjai, Tamás Nepusz, Agoston E. Eiben and Tamás Vicsek. 'Optimized flocking of autonomous drones in confined environments'. In: *Science Robotics* 3.20 (July 2018), eaat3536. DOI: 10.1126/scirobotics.aat3536. (Visited on 11/06/2024) (cit. on p. 32).

[77] K. N. McGuire, C. De Wagter, K. Tuyls, H. J. Kappen and G. C. H. E. De Croon. 'Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment'. en. In: *Science Robotics* 4.35 (Oct. 2019), eaaw9710. DOI: 10.1126/scirobotics.aaw9710. (Visited on 15/09/2024) (cit. on p. 32).

[78] Fabian Schilling, Fabrizio Schiano and Dario Floreano. 'Vision-Based Drone Flocking in Outdoor Environments'. In: *IEEE Robotics and Automation Letters* 6.2 (Apr. 2021), pp. 2954–2961. DOI: 10.1109/LRA.2021.3062298. (Visited on 15/09/2024) (cit. on p. 33).

[79] Florian Berlinger, Melvin Gauci and Radhika Nagpal. 'Implicit coordination for 3D underwater collective behaviors in a fish-inspired robot swarm'. en. In: *Science Robotics* 6.50 (Jan. 2021), eabd8668. DOI: 10.1126/scirobotics.abd8668. (Visited on 15/09/2024) (cit. on p. 33).

[80] I. Slavkov, D. Carrillo-Zapata, N. Carranza, X. Diego, F. Jansson, J. Kaandorp, S. Hauert and J. Sharpe. 'Morphogenesis in robot swarms'. en. In: *Science Robotics* 3.25 (Dec. 2018), eaau9178. DOI: 10.1126/scirobotics.aau9178. (Visited on 15/09/2024) (cit. on p. 33).

[81] Michael Rubenstein, Alejandro Cornejo and Radhika Nagpal. 'Programmable self-assembly in a thousand-robot swarm'. en. In: *Science* 345.6198 (Aug. 2014), pp. 795–799. DOI: 10.1126/science.1254295. (Visited on 15/09/2024) (cit. on p. 33).

[82] 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*. 2014, pp. 67–72. DOI: 10.1109/ICCW.2014.6881174 (cit. on p. 33).

[83] 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*. IEEE, 2018, pp. 1–9 (cit. on p. 33).

[84] 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'. In: *IEEE Control Systems Magazine* 26.3 (2006), pp. 56–69 (cit. on p. 33).

[85] Shao-cheng Wang and Ahmed Helmy. 'Performance Limits and Analysis of Contention-based IEEE 802.11 MAC'. In: *IEEE Conference on Local Computer Networks*. 2006, pp. 418–425. DOI: 10.1109/LCN.2006.322129 (cit. on p. 33).

[86] 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*. 2018, pp. 1–5. DOI: 10.1109/ICPEICES.2018.8897289 (cit. on p. 33).

[87] Jennifer Gielis and Amanda Prorok. 'Improving 802.11p for Delivery of Safety-Critical Navigation Information in Robot-to-Robot Communication Networks'. In: *IEEE Communications Magazine* 59.1 (2021), pp. 16–21. DOI: 10.1109/MCOM.001.2000545 (cit. on p. 33).

[88] 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*. 2019, pp. 1952–1958. DOI: 10.1109/SSCI44817.2019.9002704 (cit. on p. 33).

[89] Tobias Kronauer, Joshwa Pohlmann, Maximilian Matthe, Till Smejkal and Gerhard Fettweis. *Latency Analysis of ROS2 Multi-Node Systems*. 2021 (cit. on p. 33).

[90] Ekaterina Tolstaya, Fernando Gama, James Paulos, George Pappas, Vijay Kumar and Alejandro Ribeiro. 'Learning Decentralized Controllers for Robot Swarms with Graph Neural Networks'. In: *Proceedings of the Conference on Robot Learning*. Ed. by Leslie Pack Kaelbling, Danica Kragic and Komei Sugiura. Vol. 100. Proceedings of Machine Learning Research. PMLR, Nov. 2020, pp. 671–682 (cit. on pp. 33, 41, 61).

[91] Fernando Gama, Qingbiao Li, Ekaterina Tolstaya, Amanda Prorok and Alejandro Ribeiro. 'Synthesizing Decentralized Controllers With Graph Neural Networks and Imitation Learning'. In: *IEEE Transactions on Signal Processing* 70 (2022), pp. 1932–1946. DOI: 10.1109/TSP.2022.3166401 (cit. on p. 33).

[92] Tsun-Hsuan Wang, Sivabalan Manivasagam, Ming Liang, Bin Yang, Wenyuan Zeng and Raquel Urtasun. 'V2VNet: Vehicle-to-Vehicle Communication for Joint Perception and Prediction'. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox and Jan-Michael Frahm. Cham: Springer International Publishing, 2020, pp. 605–621 (cit. on p. 34).

[93] Yang Zhou, Jiuhong Xiao, Yue Zhou and Giuseppe Loianno. 'Multi-Robot Collaborative Perception With Graph Neural Networks'. In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 2289–2296. DOI: 10.1109/LRA.2022.3141661 (cit. on pp. 34, 97).

[94] 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: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 4900–4904 (cit. on pp. 34, 86).

[95] Lynxmotion A4WD3. *Lynxmotion A4WD3*. 2024. URL: https://www.lynxmotion.com/a4wd3-rugged-rovers/ (cit. on pp. 34, 35).

[96] Philipp Foehn, Elia Kaufmann, Angel Romero, Robert Penicka, Sihao Sun, Leonard Bauersfeld, Thomas Laengle, Giovanni Cioffi, Yunlong Song, Antonio Loquercio et al. 'Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight'. In: *Science robotics* 7.67 (2022), eabl6259 (cit. on pp. 34, 35).

[97] Gilles Caprari and Roland Siegwart. 'Design and control of the mobile micro robot alice'. In: *Proceedings of the 2nd International Symposium on Autonomous Minirobots for Research and Edutainment, AMiRE 2003: 18-20 February 2003, Brisbane, Australia*. CITI, 2003, pp. 23–32 (cit. on pp. 34, 35).

[98] Farshad Arvin, Khairulmizam Samsudin, Abdul Rahman Ramli et al. 'Development of a miniature robot for swarm robotic application'. In: *International Journal of Computer and Electrical Engineering* 1.4 (2009), pp. 436–442 (cit. on pp. 34, 35).

[99] Inkyu Sa, Mina Kamel, Michael Burri, Michael Bloesch, Raghav Khanna, Marija Popović, Juan Nieto and Roland Siegwart. 'Build your own visual-inertial drone: A cost-effective and open-source autonomous drone'. In: *IEEE Robotics & Automation Magazine* 25.1 (2017), pp. 89–103 (cit. on p. 34).

[100] Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg and Evangelos A. Theodorou. 'Aggressive driving with model predictive path integral control'. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1433–1440. DOI: 10.1109/ICRA.2016.7487277 (cit. on pp. 34, 35, 63).

[101] J Gonzales, F Zhang, K Li and F Borrelli. 'Autonomous drifting with onboard sensors'. In: *Advanced Vehicle Control*. CRC Press, 2016, pp. 133–138 (cit. on pp. 34, 35).

[102] Clearpath Robotics Inc. *BOXER: INDOOR ROBOTIC PLATFORM*. Tech. rep. Clearpath Robotics Inc., 2022 (cit. on pp. 34, 35).

[103] Ayberk Özgür, Séverin Lemaignan, Wafa Johal, Maria Beltran, Manon Briod, Léa Pereyre, Francesco Mondada and Pierre Dillenbourg. 'Cellulo: Versatile Handheld Robots for Education'. In: *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*. HRI '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 119–127. DOI: 10.1145/2909824.3020247 (cit. on pp. 34, 35).

[104] Hanlin Wang and Michael Rubenstein. 'Shape Formation in Homogeneous Swarms Using Local Task Swapping'. In: *IEEE Transactions on Robotics* 36.3 (2020), pp. 597–612. DOI: 10.1109/TRO.2020.2967656 (cit. on p. 34).

[105] Farshad Arvin, John C. Murray, Licheng Shi, Chun Zhang and Shigang Yue. 'Development of an autonomous micro robot for swarm robotics'. In: *2014 IEEE International Conference on Mechatronics and Automation*. 2014, pp. 635–640. DOI: 10.1109/ICMA.2014.6885771 (cit. on pp. 34, 35).

[106] Hangfa Compass Q2. *Hangfa Compass Q2*. 2024. URL: http://hangfa.com/EN/robot/CompassQ2.html (cit. on pp. 34, 35).

[107] Wojciech Giernacki, Mateusz Skwierczyński, Wojciech Witwicki, Paweł Wroński and Piotr Kozierski. 'Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering'. In: *22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE, 2017, pp. 37–42 (cit. on p. 34).

[108] Bharathan Balaji et al. 'DeepRacer: Autonomous Racing Platform for Experimentation with Sim2Real Reinforcement Learning'. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 2746–2754. DOI: 10.1109/ICRA40945.2020.9197465 (cit. on pp. 34, 35).

[109] DirectDriveTech. *Diablo: World's First Direct-Drive Self-Balancing Wheeled-Leg Robot*. 2024. URL: https://shop.directdrive.com/products/diablo-world-s-first-direct-drive-self-balancing-wheeled-leg-robot (cit. on pp. 34, 35).

[110] Clearpath Robotics Inc. *DINGO: INDOOR ROBOTIC PLATFORM*. Tech. rep. Clearpath Robotics Inc., 2019 (cit. on pp. 34, 35).

[111] Liam Paull et al. 'Duckietown: An open, inexpensive and flexible platform for autonomy education and research'. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 1497–1504. DOI: 10.1109/ICRA.2017.7989179 (cit. on pp. 34, 35).

[112] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stephane Magnenat, Jean-Christophe Zufferey, Dario Floreano and Alcherio Martinoli. 'The e-puck, a robot designed for education in engineering'. In: *Proceedings of the 9th conference on autonomous robot systems and competitions*. Vol. 1. IPCB: Instituto Politécnico de Castelo Branco, 2009, pp. 59–65 (cit. on pp. 34, 35).

[113] Kartik Mohta, Michael Watterson, Yash Mulgaonkar, Sikang Liu, Chao Qu, Anurag Makineni, Kelsey Saulnier, Ke Sun, Alex Zhu, Jeffrey Delmerico et al. 'Fast, autonomous flight in GPS-denied and cluttered environments'. In: *Journal of Field Robotics* 35.1 (2018), pp. 101–120 (cit. on p. 34).

[114] Giuseppe Loianno, Chris Brunner, Gary McGrath and Vijay Kumar. 'Estimation, Control, and Planning for Aggressive Flight With a Small Quadrotor With a Single Camera and IMU'. In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 404–411. DOI: 10.1109/LRA.2016.2633290 (cit. on p. 34).

[115] Agilex Hunter. *Agilex Hunter Website*. 2024. URL: https://global.agilex.ai/chassis/9 (cit. on pp. 34, 35).

[116] Clearpath Robotics Inc. *JACKAL Unmannaged Ground Vehicle: Technical Specifications*. Tech. rep. Clearpath Robotics Inc., 2020 (cit. on pp. 34, 35, 63).

[117] Serge Kernbach, Ronald Thenius, Olga Kernbach and Thomas Schmickl. 'Re-embodiment of honeybee aggregation behavior in an artificial micro-robotic system'. In: *Adaptive Behavior* 17.3 (2009), pp. 237–259 (cit. on pp. 34, 35).

[118] RoboWorks Mecabot Pro. *RoboWorks Mecabot Pro*. 2024. URL: https://www.roboworks.net/store/p/mecabotpro (cit. on pp. 34, 35).

[119] Jorge M Soares, Inaki Navarro and Alcherio Martinoli. 'The Khepera IV mobile robot: performance evaluation, sensory data and software toolbox'. In: *Robot 2015: Second Iberian Robotics Conference: Advances in Robotics, Volume 1*. Springer, 2016, pp. 767–781 (cit. on pp. 34, 35).

[120] RoboWorks Mecabot X. *RoboWorks Mecabot X*. 2024. URL: https://www.roboworks.net/store/p/mecabot-x (cit. on pp. 34, 35).

[121] Nicholas Hyldmar, Yijun He and Amanda Prorok. 'A Fleet of Miniature Cars for Experiments in Cooperative Driving'. In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 3238–3244. DOI: 10.1109/ICRA.2019.8794445 (cit. on pp. 34, 35).

[122] Sertac Karaman, Ariel Anders, Michael Boulet, Jane Connor andKenneth Gregson, Winter J. Guerra, Owen Guldner, Mubarik Mohamoudand Brian Plancher, Robert Shin and John Vivilecchia. 'Project-based, Collaborative, Algorithmic Robotics for High SchoolStudents: Programming Self-driving Race Cars at MIT'. In: *2017 IEEE Integrated STEM Education Conference (ISEC) (ISEC'17)*. Princeton, USA, Mar. 2017 (cit. on pp. 34, 35).

[123] Amado Antonini, Winter Guerra, Varun Murali, Thomas Sayre-McCord and Sertac Karaman. 'The blackbird dataset: A large-scale dataset for uav perception in aggressive flight'. In: *International Symposium on Experimental Robotics*. Springer, 2018, pp. 130–139 (cit. on p. 34).

[124] Farshad Arvin, Jose Espinosa, Benjamin Bird, Andrew West, Simon Watson and Barry Lennox. 'Mona: an affordable open-source mobile robot for education and research'. In: *Journal of Intelligent & Robotic Systems* 94 (2019), pp. 761–775 (cit. on pp. 34, 35).

[125] Hangfa Navigator Q2. *Hangfa Navigator Q2*. 2024. URL: http://hangfa.com/EN/robot/NavigatorQ2.html (cit. on pp. 34, 35).

[126] Alexander Liniger, Alexander Domahidi and Manfred Morari. 'Optimization-based autonomous racing of 1: 43 scale RC cars'. In: *Optimal Control Applications and Methods* 36.5 (2015), pp. 628–647 (cit. on pp. 34, 35).

[127] Sean Wilson, Ruben Gameros, Michael Sheely, Matthew Lin, Kathryn Dover, Robert Gevorkyan, Matt Haberland, Andrea Bertozzi and Spring Berman. 'Pheeno, A Versatile Swarm Robotic Research and Education Platform'. In: *IEEE Robotics and Automation Letters* 1.2 (2016), pp. 884–891. DOI: 10.1109/LRA.2016.2524987 (cit. on pp. 34, 35).

[128] James McLurkin, Adam McMullen, Nick Robbins, Golnaz Habibi, Aaron Becker, Alvin Chou, Hao Li, Meagan John, Nnena Okeke, Joshua Rykowski et al. 'A robot system design for low-cost multi-robot manipulation'. In: *2014 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2014, pp. 912–918 (cit. on pp. 34, 35).

[129] Agilex Ranger Mini. *Agilex Ranger Mini Website*. 2024. URL: https://global.agilex.ai/chassis/6 (cit. on pp. 34, 35).

[130] Clearpath Robotics Inc. *RIDGEBACK: Omnidirectional Development Platform*. Tech. rep. Clearpath Robotics Inc., 2020 (cit. on pp. 34, 35).

[131] FESTO Robotino. *FESTO Robotino*. 2024. URL: https://ip.festo-didactic.com/InfoPortal/Robotino/Overview/EN/index.html (cit. on pp. 34, 35).

[132] Matthias Faessler, Antonio Franchi and Davide Scaramuzza. 'Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories'. In: *IEEE Robotics and Automation Letters* 3.2 (2017), pp. 620–626 (cit. on p. 34).

[133] Agilex Titan. *Agilex Titan Website*. 2024. URL: https://global.agilex.ai/chassis/15 (cit. on pp. 34, 35).

[134] Turtlebot3. *TurtleBot3 Website*. 2024. URL: https://www.turtlebot.com/turtlebot3/ (cit. on pp. 34, 35, 63).

[135] Clearpath Robotics Inc. *Turtlebot 4: Robotics Learning Platform*. Tech. rep. Clearpath Robotics Inc., 2022 (cit. on pp. 34, 35).

[136] RDK X3. *RDK X3*. 2024. URL: https://category.yahboom.net/collections/r-omnidirection/products/rdk-x3-robot?variant=46378068967740 (cit. on pp. 34, 35).

[137] Rainer Bischoff, Ulrich Huggenberger and Erwin Prassler. 'KUKA youBot - a mobile manipulator for research and education'. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 1–4. DOI: 10.1109/ICRA.2011.5980575 (cit. on pp. 34, 35).

[138] Rishi Bommasani et al. *On the Opportunities and Risks of Foundation Models*. 2021. DOI: 10.48550/ARXIV.2108.07258. URL: https://arxiv.org/abs/2108.07258 (visited on 15/09/2024) (cit. on pp. 36, 97).

[139] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser and Illia Polosukhin. 'Attention is All you Need'. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett. Vol. 30. Curran Associates, Inc., 2017 (cit. on pp. 36, 99).

[140] Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding'. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423 (cit. on p. 36).

[141] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever et al. 'Language models are unsupervised multitask learners'. In: *OpenAI blog* 1.8 (2019), p. 9 (cit. on p. 36).

[142] Tom Brown et al. 'Language Models are Few-Shot Learners'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901 (cit. on p. 36).

[143] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat et al. 'Gpt-4 technical report'. In: *arXiv preprint arXiv:2303.08774* (2023) (cit. on p. 36).

[144] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski and Armand Joulin. *Emerging Properties in Self-Supervised Vision Transformers*. 2021. URL: https://arxiv.org/abs/2104.14294 (cit. on p. 36).

[145] Maxime Oquab et al. *DINOv2: Learning Robust Visual Features without Supervision*. 2023. URL: https://github.com/facebookresearch/dinov2/tree/main (cit. on pp. 36, 120).

[146] Alec Radford et al. 'Learning Transferable Visual Models From Natural Language Supervision'. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 8748–8763 (cit. on p. 36).

[147] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen and Ilya Sutskever. 'Zero-Shot Text-to-Image Generation'. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 8821–8831 (cit. on p. 36).

[148] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu and Mark Chen. 'Hierarchical text-conditional image generation with clip latents'. In: *arXiv preprint arXiv:2204.06125* 1.2 (2022), p. 3 (cit. on p. 36).

[149] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser and Björn Ommer. 'High-resolution image synthesis with latent diffusion models'. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10684–10695 (cit. on p. 36).

[150] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu et al. 'Rt-1: Robotics transformer for real-world control at scale'. In: *arXiv preprint arXiv:2212.06817* (2022) (cit. on p. 36).

[151] Anthony Brohan et al. *RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control*. 2023. URL: https://arxiv.org/abs/2307.15818 (cit. on p. 36).

[152] Moo Jin Kim et al. *OpenVLA: An Open-Source Vision-Language-Action Model*. 2024. URL: https://arxiv.org/abs/2406.09246 (cit. on p. 36).

[153] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar et al. 'Llama: Open and efficient foundation language models'. In: *arXiv preprint arXiv:2302.13971* (2023) (cit. on p. 36).

[154] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. URL: https://arxiv.org/abs/2307.09288 (cit. on p. 36).

[155] Abhimanyu Dubey et al. *The Llama 3 Herd of Models*. 2024. URL: https://arxiv.org/abs/2407.21783 (cit. on p. 36).

[156] Geoffrey Hinton, Oriol Vinyals and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. DOI: 10.48550/ARXIV.1503.02531. URL: https://arxiv.org/abs/1503.02531 (visited on 29/09/2024) (cit. on p. 36).

[157] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam and Dmitry Kalenichenko. 'Quantization and training of neural networks for efficient integer-arithmetic-only inference'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2704–2713 (cit. on pp. 36, 131).

[158] Mingsheng Long, Yue Cao, Jianmin Wang and Michael I. Jordan. 'Learning Transferable Features with Deep Adaptation Networks'. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. JMLR.org, 2015, pp. 97–105 (cit. on p. 37).

[159] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand and Victor Lempitsky. 'Domain-Adversarial Training of Neural Networks'. In: *J. Mach. Learn. Res.* 17.1 (Jan. 2016), pp. 2096–2030 (cit. on p. 37).

[160] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan and Dumitru Erhan. 'Domain Separation Networks'. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, pp. 343–351 (cit. on p. 37).

[161] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan and Dilip Krishnan. 'Unsupervised Pixel-Level Domain Adaptation With Generative Adversarial Networks'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017 (cit. on p. 37).

[162] Zili Yi, Hao Zhang, Ping Tan and Minglun Gong. 'DualGAN: Unsupervised Dual Learning for Image-To-Image Translation'. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017 (cit. on p. 37).

[163] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Pieter Abbeel, Sergey Levine, Kate Saenko and Trevor Darrell. 'Adapting Deep Visuomotor Representations with Weak Pairwise Constraints'. en. In: *Algorithmic Foundations of Robotics XII*. Ed. by Ken Goldberg, Pieter Abbeel, Kostas Bekris and Lauren Miller. Vol. 13. Cham: Springer International Publishing, 2020, pp. 688–703. DOI: 10.1007/978-3-030-43089-4_44. (Visited on 25/08/2024) (cit. on p. 37).

[164] Gregory J Stein and Nicholas Roy. 'Genesis-rt: Generating synthetic images for training secondary real-world tasks'. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7151–7158 (cit. on p. 37).

[165] Fereshteh Sadeghi and Sergey Levine. 'CAD2RL: Real Single-Image Flight Without a Single Real Image'. In: *Proceedings of Robotics: Science and Systems*. Cambridge, Massachusetts, July 2017. DOI: 10.15607/RSS.2017.XIII.034 (cit. on p. 37).

[166] Stephen James, Andrew J Davison and Edward Johns. 'Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task'. In: *Conference on Robot Learning*. PMLR, 2017, pp. 334–343 (cit. on p. 37).

[167] Jan Matas, Stephen James and Andrew J Davison. 'Sim-to-real reinforcement learning for deformable object manipulation'. In: *Conference on Robot Learning*. PMLR, 2018, pp. 734–743 (cit. on p. 37).

[168] 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: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 4243–4250 (cit. on p. 37).

[169] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J Pal and Liam Paull. 'Active domain randomization'. In: *Conference on Robot Learning*. PMLR, 2020, pp. 1162–1176 (cit. on p. 37).

[170] 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'. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1148–1155 (cit. on pp. 37, 38, 90).

[171] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. J. Burges, L. Bottou and K. Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012 (cit. on p. 37).

[172] Linxi Fan, Guanzhi Wang, De-An Huang, Zhiding Yu, Li Fei-Fei, Yuke Zhu and Animashree Anandkumar. 'SECANT: Self-Expert Cloning for Zero-Shot Generalization of Visual Policies'. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 3088–3099 (cit. on p. 37).

[173] Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun and Davide Scaramuzza. 'Deep drone racing: From simulation to reality with domain randomization'. In: *IEEE Transactions on Robotics* 36.1 (2019), pp. 1–14 (cit. on p. 37).

[174] Ashish Kumar, Zipeng Fu, Deepak Pathak and Jitendra Malik. 'Rma: Rapid motor adaptation for legged robots'. In: *Robotics: Science and Systems*. 2021 (cit. on p. 37).

[175] Igor Mordatch, Kendall Lowrey and Emanuel Todorov. 'Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids'. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 5307–5314 (cit. on p. 37).

[176] Rika Antonova, Silvia Cruciani, Christian Smith and Danica Kragic. 'Reinforcement learning for pivoting task'. In: *arXiv preprint arXiv:1703.00472* (2017) (cit. on p. 37).

[177] Wenhao Yu, Jie Tan, C Karen Liu and Greg Turk. 'Preparing for the unknown: Learning a universal policy with online system identification'. In: *arXiv preprint arXiv:1702.02453* (2017) (cit. on p. 37).

[178] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba and Pieter Abbeel. 'Sim-to-real transfer of robotic control with dynamics randomization'. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810 (cit. on p. 37).

[179] Chieko Sarah Imai, Minghao Zhang, Yuchen Zhang, Marcin Kierebiński, Ruihan Yang, Yuzhe Qin and Xiaolong Wang. 'Vision-guided quadrupedal locomotion in the wild with multi-modal delay randomization'. In: *2022 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2022, pp. 5556–5563 (cit. on p. 37).

[180] Matthias Müller, Alexey Dosovitskiy, Bernard Ghanem and Vladlen Koltun. 'Driving policy transfer via modularity and abstraction'. In: *Conference on Robot Learning*. PMLR, 2018 (cit. on p. 38).

[181] 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: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, June 2019, pp. 12619–12629. DOI: 10.1109/CVPR.2019.01291 (cit. on p. 38).

[182] 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)*. 2017, pp. 5967–5976 (cit. on p. 38).

[183] 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 (ICCV)*. 2017, pp. 2242–2251 (cit. on p. 38).

[184] 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. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 1928–1930 (cit. on p. 38).

[185] Amanda Prorok. *Graph Neural Networks for Learning Robot Team Coordination. Federated AI for Robotics Workshop*. Tech. rep. IJCAI-ECAI/ICML/AAMAS 2018, 2018 (cit. on p. 41).

[186] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan and Pieter Abbeel. 'High-Dimensional Continuous Control Using Generalized Advantage Estimation'. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2016 (cit. on pp. 42, 43).

[187] Philipp Moritz et al. 'Ray: A Distributed Framework for Emerging AI Applications'. In: *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*. OSDI'18. USA: USENIX Association, 2018, pp. 561–577 (cit. on p. 46).

[188] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan and Ion Stoica. 'RLlib: Abstractions for Distributed Reinforcement Learning'. In: *International Conference on Machine Learning (ICML)*. 2018 (cit. on p. 46).

[189] Enric Galceran and Marc Carreras. 'A survey on coverage path planning for robotics'. In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1258–1276. DOI: https://doi.org/10.1016/j.robot.2013.09.004 (cit. on pp. 47, 50).

[190] Matthew D. Zeiler and Rob Fergus. 'Visualizing and Understanding Convolutional Networks'. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet, Tomas Pajdla, Bernt Schiele and Tinne Tuytelaars. Cham: Springer International Publishing, 2014, pp. 818–833 (cit. on p. 54).

[191] Misa Ogura and Ravi Jain. *FlashTorch: Visualization toolkit for neural networks in PyTorch*. 2020. URL: https://github.com/MisaOgura/flashtorch (cit. on p. 54).

[192] Ryan Kortvelesy and Amanda Prorok. 'ModGNN: Expert Policy Approximation in Multi-Agent Systems with a Modular Graph Neural Network Architecture'. In: *IEEE International Conference on Robotics and Automation*. IEEE, 2021 (cit. on p. 61).

[193] 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'. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 1120–1127 (cit. on p. 61).

[194] 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*. IEEE, 2016, pp. 92–99 (cit. on p. 61).

[195] James Stephan, Jonathan Fink, Vijay Kumar and Alejandro Ribeiro. 'Concurrent control of mobility and communication in multirobot systems'. In: *IEEE Transactions on Robotics* 33.5 (2017), pp. 1248–1254 (cit. on p. 61).

[196] 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*. IEEE, 2014, pp. 3753–3760 (cit. on p. 61).

[197] Matteo Bettini, Ryan Kortvelesy, Jan Blumenkamp and Amanda Prorok. 'VMAS: A Vectorized Multi-Agent Simulator for Collective Robot Learning'. In: *Proceedings of the 16th International Symposium on Distributed Autonomous Robotic Systems*. DARS '22. Springer, 2022 (cit. on pp. 62, 76).

[198] RoboMaster Organizing Committee (RMOC). *RoboMaster 2023 University League Rules Manual*. Tech. rep. T2, 22F, DJI Sky City, No. 55 Xianyuan Road, Nanshan District, Shenzen, China: DJI, Nov. 2022, p. 73 (cit. on p. 63).

[199] DJI RoboMaster S1. *DJI RoboMaster S1 Website*. 2024. URL: https://www.dji.com/uk/robomaster-s1 (cit. on p. 64).

[200] 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*. Vol. 3. Kobe, Japan, 2009, p. 5 (cit. on p. 65).

[201] Open Robotics. *ROS 2 Design*. 2021. URL: https://design.ros2.org/ (cit. on p. 65).

[202] eProsima. *Fast-DDS*. 2020. URL: https://github.com/eProsima/Fast-DDS (cit. on p. 68).

[203] Mayank Mittal et al. 'Orbit: A Unified Simulation Framework for Interactive Robot Learning Environments'. In: *IEEE Robotics and Automation Letters* 8.6 (2023), pp. 3740–3747. DOI: 10.1109/LRA.2023.3270034 (cit. on p. 76).

[204] Matteo Bettini, Ajay Shankar and Amanda Prorok. 'Heterogeneous Multi-Robot Reinforcement Learning'. In: *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '23. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2023, pp. 1485–1494 (cit. on p. 76).

[205] Matteo Bettini, Amanda Prorok and Vincent Moens. 'BenchMARL: Benchmarking Multi-Agent Reinforcement Learning'. In: *arXiv preprint arXiv:2312.01472* (2023) (cit. on p. 77).

[206] Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis and Vincent Moens. 'TorchRL: A data-driven decision-making library for PyTorch'. In: *The Twelfth International Conference on Learning Representations*. 2024 (cit. on p. 77).

[207] Piotr W. Mirowski et al. 'Learning to Navigate in Complex Environments'. In: *ArXiv* abs/1611.03673 (2017) (cit. on p. 81).

[208] Andreas Kolling and Stefano Carpin. 'Surveillance Strategies for Target Detection with Sweep Lines'. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009, pp. 5821–5827 (cit. on p. 81).

[209] Peter Corke, Ron Peterson and Daniela Rus. 'Networked Robots: Flying Robot Navigation using a Sensor Net'. In: *International Symposium on Robotics Research*. 2005, pp. 234–243 (cit. on p. 81).

[210] Zendai Kashino, Goldie Nejat and Beno Benhabib. 'A Hybrid Strategy for Target Search Using Static and Mobile Sensors'. In: *IEEE Transactions on Cybernetics* 50.2 (2020), pp. 856–868 (cit. on p. 81).

[211] Mohak Bhardwaj, Sanjiban Choudhury and Sebastian Scherer. 'Learning Heuristic Search via Imitation'. In: *Conference on Robot Learning*. PMLR, 2017, pp. 271–280 (cit. on p. 83).

[212] Stéphane Ross and J. Andrew Bagnell. 'Reinforcement and Imitation Learning via Interactive No-Regret Learning'. In: *ArXiv* abs/1406.5979 (2014) (cit. on p. 83).

[213] Stephane Ross and Drew Bagnell. 'Efficient Reductions for Imitation Learning'. In: *International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, May 2010, pp. 661–668 (cit. on p. 84).

[214] Dave Coleman. *Lee's O (n2 log n) Visibility Graph Algorithm Implementation and Analysis*. 2012 (cit. on p. 85).

[215] 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 (cit. on p. 86).

[216] Syeda Selina Akter and Lawrence B. Holder. 'Improving IoT Predictions through the Identification of Graphical Features'. In: *Sensors* 19.15 (2019), p. 3250 (cit. on p. 86).

[217] 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*. Vol. 33. AAAI Press, 2019, pp. 4602–4609 (cit. on p. 87).

[218] O. Michel. 'Webots: Professional Mobile Robot Simulation'. In: *Journal of Advanced Robotics Systems* 1.1 (2004), pp. 39–42 (cit. on p. 89).

[219] Yong Koo Hwang and Narendra Ahuja. 'A Potential Field Approach to Path Planning'. In: *IEEE Transactions on Robotics and Automation* 8.1 (1992), pp. 23–32 (cit. on p. 89).

[220] Zhengyou Zhang. 'A Flexible New Technique for Camera Calibration'. In: *IEEE Transactions on pattern analysis and machine intelligence* 22.11 (2000), pp. 1330–1334 (cit. on p. 90).

[221] 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'. In: *arXiv preprint arXiv:1807.06757* (2018) (cit. on p. 91).

[222] Davide Scaramuzza and Friedrich Fraundorfer. 'Visual Odometry [Tutorial]'. In: *IEEE Robotics & Automation Magazine* 18.4 (2011), pp. 80–92. DOI: 10.1109/MRA.2011.943233 (cit. on p. 97).

[223] Martin Saska, Jan Vakula and Libor Přeućil. 'Swarms of micro aerial vehicles stabilized under a visual relative localization'. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 3570–3575. DOI: 10.1109/ICRA.2014.6907374 (cit. on p. 97).

[224] Oscar De Silva, George K. I. Mann and Raymond G. Gosine. 'An Ultrasonic and Vision-Based Relative Positioning Sensor for Multirobot Localization'. In: *IEEE Sensors Journal* 15.3 (2015), pp. 1716–1726. DOI: 10.1109/JSEN.2014.2364684 (cit. on p. 97).

[225] Viktor Walter, Martin Saska and Antonio Franchi. 'Fast Mutual Relative Localization of UAVs using Ultraviolet LED Markers'. In: *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2018, pp. 1217–1226. DOI: 10.1109/ICUAS.2018.8453331 (cit. on p. 97).

[226] Johannes L. Schönberger and Jan-Michael Frahm. 'Structure-from-Motion Revisited'. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4104–4113. DOI: 10.1109/CVPR.2016.445 (cit. on p. 97).

[227] Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, José M. M. Montiel and Juan D. Tardós. 'ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM'. In: *IEEE Transactions on Robotics* 37.6 (2021), pp. 1874–1890. DOI: 10.1109/TRO.2021.3075644 (cit. on p. 97).

[228] Akmaral Moldagalieva and Wolfgang Hönig. *Virtual Omnidirectional Perception for Downwash Prediction within a Team of Nano Multirotors Flying in Close Proximity*. 2023. URL: https://arxiv.org/abs/2303.03898 (cit. on p. 97).

[229] Paola Torrico Morón, Sahar Salimpour, Lei Fu, Xianjia Yu, Jorge Peña Queralta and Tomi Westerlund. *Benchmarking UWB-Based Infrastructure-Free Positioning and Multi-Robot Relative Localization: Dataset and Characterization*. 2023. URL: https://arxiv.org/abs/2305.08532 (cit. on p. 97).

[230] Iaroslav Melekhov, Juha Ylioinas, Juho Kannala and Esa Rahtu. 'Relative Camera Pose Estimation Using Convolutional Neural Networks'. In: *Advanced Concepts for Intelligent Vision Systems*. Ed. by Jacques Blanc-Talon, Rudi Penne, Wilfried Philips, Dan Popescu and Paul Scheunders. Cham: Springer International Publishing, 2017, pp. 675–687 (cit. on p. 97).

[231] Praveen Kumar Rajendran, Sumit Mishra, Luiz Felipe Vecchietti and Dongsoo Har. 'RelMobNet: End-to-End Relative Camera Pose Estimation Using a Robust Two-Stage Training'. In: *Computer Vision – ECCV 2022 Workshops*. Ed. by Leonid Karlinsky, Tomer Michaeli and Ko Nishino. Cham: Springer Nature Switzerland, 2023, pp. 238–252 (cit. on p. 97).

[232] Chenhao Yang, Yuyi Liu and Andreas Zell. 'RCPNet: Deep-Learning based Relative Camera Pose Estimation for UAVs'. In: *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2020, pp. 1085–1092. DOI: 10.1109/ICUAS48674.2020.9214000 (cit. on p. 97).

[233] Chenhao Yang, Yuyi Liu and Andreas Zell. 'Learning-based Camera Relocalization with Domain Adaptation via Image-to-Image Translation'. In: *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2021, pp. 1047–1054. DOI: 10.1109/ICUAS51884.2021.9476673 (cit. on p. 97).

[234] Ruojin Cai, Bharath Hariharan, Noah Snavely and Hadar Averbuch-Elor. 'Extreme Rotation Estimation using Dense Correlation Volumes'. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021 (cit. on p. 97).

[235] Eduardo Arnold, Jamie Wynn, Sara Vicente, Guillermo Garcia-Hernando, Aron Monszpart, Victor Adrian Prisacariu, Daniyar Turmukhambetov and Eric Brachmann. 'Map-free Visual Relocalization: Metric Pose Relative to a Single Image'. In: *ECCV*. 2022 (cit. on p. 97).

[236] Chris Rockwell, Justin Johnson and David F. Fouhey. 'The 8-Point Algorithm as an Inductive Bias for Relative Pose Prediction by ViTs'. In: *2022 International Conference on 3D Vision (3DV)*. 2022, pp. 1–11. DOI: 10.1109/3DV57658.2022.00028 (cit. on p. 97).

[237] Karl Weiss, Taghi M Khoshgoftaar and DingDing Wang. 'A survey of transfer learning'. In: *Journal of Big data* 3 (2016), pp. 1–40 (cit. on pp. 99, 101).

[238] Alexey Dosovitskiy et al. 'An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale'. In: *International Conference on Learning Representations*. 2021 (cit. on p. 99).

[239] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei. 'ImageNet: A large-scale hierarchical image database'. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848 (cit. on p. 100).

[240] Andrew Szot et al. 'Habitat 2.0: Training Home Assistants to Rearrange their Habitat'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021 (cit. on p. 102).

[241] Santhosh Kumar Ramakrishnan et al. 'Habitat-Matterport 3D Dataset (HM3D): 1000 Large-scale 3D Environments for Embodied AI'. In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2021 (cit. on p. 102).

[242] Alex Kendall and Yarin Gal. 'What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?' In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett. Vol. 30. Curran Associates, Inc., 2017 (cit. on p. 103).

[243] Yarin Gal and Zoubin Ghahramani. 'Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning'. In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, June 2016, pp. 1050–1059 (cit. on p. 103).

[244] Balaji Lakshminarayanan, Alexander Pritzel and Charles Blundell. 'Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles'. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett. Vol. 30. Curran Associates, Inc., 2017 (cit. on p. 103).

[245] D.A. Nix and A.S. Weigend. 'Estimating the mean and variance of the target probability distribution'. In: *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*. Vol. 1. 1994, 55–60 vol.1. DOI: 10.1109/ICNN.1994.374138 (cit. on p. 103).

[246] Valentin Peretroukhin, Matthew Giamou, W. Nicholas Greene, David Rosen, Jonathan Kelly and Nicholas Roy. 'A Smooth Representation of Belief over SO(3) for Deep Rotation Learning with Uncertainty'. In: *Proceedings of Robotics: Science and Systems*. Corvalis, Oregon, USA, July 2020. DOI: 10.15607/RSS.2020.XVI.007 (cit. on p. 103).

[247] Carole H Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin and M Jorge Cardoso. 'Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations'. In: *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3*. Springer, 2017, pp. 240–248 (cit. on p. 104).

[248] Shruti Jadon. 'A survey of loss functions for semantic segmentation'. In: *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. 2020, pp. 1–7. DOI: 10.1109/CIBCB48159.2020.9277638 (cit. on p. 104).

[249] William J Youden. 'Index for rating diagnostic tests'. In: *Cancer* 3.1 (1950), pp. 32–35 (cit. on p. 105).

[250] Philipp Lindenberger, Paul-Edouard Sarlin and Marc Pollefeys. 'LightGlue: Local Feature Matching at Light Speed'. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2023, pp. 17627–17638 (cit. on p. 110).

[251] Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary Bradski. 'ORB: An efficient alternative to SIFT or SURF'. In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544 (cit. on p. 110).

[252] Alexander Kirillov et al. 'Segment Anything'. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2023, pp. 4015–4026 (cit. on p. 120).

[253] 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*. IEEE, 2021, pp. 217–223 (cit. on p. 130).

[254] 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*. IEEE, 2014, pp. 1301–1307 (cit. on p. 130).

[255] Christopher Mei and Patrick Rives. 'Single View Point Omnidirectional Camera Calibration from Planar Grids'. In: *IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 3945–3950 (cit. on p. 130).

[256] NVIDIA. *TensorRT*. 2024. URL: https://github.com/NVIDIA/TensorRT (cit. on p. 131).

Glossary

B

BEV bird's-eye-view. 96, 97, 99–106, 108, 110, 112

C

CAN Controller area network. 62, 123, 124
CNN Convolutional neural network. 11, 18, 19, 44, 54–57
COMA Counterfactual multi-agent policy gradients. 29, 43
CPP Coverage path planning. 7, 58, 117

D

DDPG Deep deterministic policy gradient. 29
Dec-POMDP Decentralized partially observable markov decision process. 21, 22
DQN Deep Q-learning. 25, 26

G

GNN Graph neural network. iii, iv, 4, 5, 7–11, 19, 20, 31, 32, 41, 42, 44, 51, 55–61, 63, 67, 70, 72–76, 80, 84–86, 89, 93, 100, 117

L

LLM Large language model. 34, 35

M

mAP mean average precision. 56
MAPF Multi-agent path planning. 7, 32
MARL Multi-agent reinforcement learning. 11, 27, 29
MAS Multi-agent system. 11, 12, 22, 29
MDP Markov decision process. 21, 22, 25, 81, 85
MLP Multi layer perceptron. 16, 55, 70
MRS Multi-robot system. iii, iv, 3–8, 10, 11, 29, 30, 34, 41, 44, 58, 59, 61, 63, 66, 68, 74, 76, 79, 95, 96, 117–119

N

NLP Natural language processing. 34
NN Neural network. 25, 26, 29, 41, 53, 54, 117, 123

P

POMDP Partially observable markov decision process. 3, 21, 22
POMG Partially observable markov game. 21, 22, 42
pp percentage points. 72, 73, 76
PPO Proximal policy optimization. 7, 26, 27

R

RL Reinforcement learning. iii, 3, 5, 11, 21, 23–25, 27, 29, 36, 42, 59, 65

S

SPL Success weighted by path length. 89–93

T

TRPO Trust region policy optimization. 26, 29

TSP Traveling salesperson problem. 3

V

VLA Vision, language and action model. 34, 35

VLM Vision and language model. 34

VPG Vanilla policy gradient. 42–45

VRP Vehicle routing problem. 3