ULB ECOLE POLYTECHNIQUE DE BRUXELLES

Resilient coordination in aerial robot swarms:

self-organizing hierarchy, reconfigurable network topologies and proactive fault-tolerance

Thesis presented by Sinan OĞUZ

in fulfilment of the requirements of the
Ph.D. Degree in Engineering Sciences and Technology
("Docteur en Sciences de l'Ingénieur et Technologie")
Année académique 2025-2026

Supervisor:
Professor Marco DORIGO
IRIDIA - Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle

Co-supervisors:
Professor Emanuele GARONE
SAAS - Service d'Automatique et d'Analyse des Systèmes
Dr. Mary Katherine HEINRICH
IRIDIA - Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle

Resilient coordination in aerial robot swarms:

self-organizing hierarchy, reconfigurable network topologies, and proactive fault tolerance

Sinan OĞUZ

2025

Statement of Originality

This thesis presents an original work that has never been submitted to the Université libre de Bruxelles or to any other institution for the award of a doctoral degree. Some parts of this thesis are based on peer-reviewed articles that the author, together with co-authors, has published in the scientific literature.

The dagger symbol (†) signifies that the authors contributed equally and should be regarded as co-first authors.

Chapter 3, *S-drone: Aerial Robotic Platform*, is based on:

- **Oğuz, S.**, Heinrich, M.K., Allwright, M., Zhu, W., Wahby, M., Garone, E. and Dorigo, M. (2024). An Open-Source UAV Platform for Swarm Robotics Research: Using Cooperative Sensor Fusion for Inter-Robot Tracking. *IEEE Access*, 12, 43378–43395.

- With the preliminary results having been presented at the following regional conference:
  Oğuz, S., Heinrich, M.K., Allwright, M., Zhu, W., Wahby, M., Dorigo, M., and Garone, E. (2022). A novel aerial robotic swarm hardware. In *Proceedings of the 41st Benelux Meeting on Systems and Control: Book of Abstracts* (p. 96). Brussels, Belgium.

Chapter 4, *Self-organizing Hierarchy for Robot Swarms*, is based on:

- Zhu, W.†, **Oğuz, S.†**, Heinrich, M.K.†, Allwright, M., Wahby, M., Christensen, A.L., Garone, E. & Dorigo, M. (2024). Self-organizing nervous systems for robot swarms. *Science Robotics*, 9(96), eadl5161.

Chapter 5, *Reconfigurable Frameworks for Self-organized Hierarchy*, is based on:

- Zhang, Y., **Oğuz, S.**, Wang, S., Garone, E., Wang, X., Dorigo, M. and Heinrich, M.K. (2023). Self-reconfigurable hierarchical frameworks for formation control of robot swarms. *IEEE Transactions on Cybernetics*, 54(1), 87–100.

Chapter 6, *Proactive Fault Tolerance in Self-organized Hierarchy*, is based on a manuscript that is currently under review:

- **Oğuz, S.**, Garone, E., Dorigo, M., and Heinrich, M.K. (2025). Proactive–reactive detection and mitigation of intermittent faults in robot swarms. Submitted to *IEEE Transactions on Robotics*. Pre-print available at arXiv:2509.19246.

The author also contributed to the following research project not presented in this thesis:

- Zhu, W., Allwright, M., Heinrich, M.K., **Oğuz, S.**, Christensen, A.L. and Dorigo, M. (2020). Formation control of UAVs and mobile robots using self-organized communication topologies. In *Swarm Intelligence – Proceedings of ANTS 2020 – 12th International Conference* (pp. 306–314). Springer.

Sinan OĞUZ
2025

Thesis Committee

- **Mauro BIRATTARI, Ph.D.**
  Université Libre de Bruxelles (ULB), Brussels, Belgium
- **Marco DORIGO, Ph.D.**
  Université Libre de Bruxelles (ULB), Brussels, Belgium
- **Emanuele GARONE, Ph.D.**
  Université Libre de Bruxelles (ULB), Brussels, Belgium
- **Mary Katherine HEINRICH, Ph.D.**
  Université Libre de Bruxelles (ULB), Brussels, Belgium
- **Enrico NATALIZIO, Ph.D.**
  Université de Lorraine, Nancy, France
- **Erol ŞAHİN, Ph.D.**
  Middle East Technical University (METU), Ankara, Türkiye

Abstract

In recent years, swarms of aerial robots have attracted growing attention in both scientific research and practical applications, driven by their potential to collectively perform tasks that would otherwise overwhelm the limitations of individual robots. Unlike a single unit acting in isolation, a swarm leverages coordination—each robot makes decisions based on simple rules and local perceptions, yet together they can generate complex, coordinated behaviors. This emergent cooperation enables the group to tackle diverse tasks with adaptability and resilience.

While robot swarms demonstrate remarkable potential for collective intelligence, reliably translating this promise into practice presents sizable technical challenges. For aerial swarms specifically, these challenges include: (i) the scarcity of open, swarm-ready UAV platforms that can support research into new aerial swarm behaviors in laboratory environments; (ii) the difficulty of developing self-organized UAV coordination algorithms that are both provably reliable—with formal guarantees for stability, convergence, and robustness—and practical to deploy under real-world constraints such as limited computation, communication, and energy; and (iii) a lack of existing self-organized approaches to handle intermittent sensing and communication disturbances in aerial swarms—including temporally correlated dropouts, delays, and noise—that, while initially non-catastrophic, can cumulatively degrade coordination quality and system-level performance and may eventually manifest as permanent faults, thereby compromising long-term swarm reliability.

This thesis advances resilient coordination in aerial robot swarms through contributions spanning hardware, theory, and algorithms. The first contribution is the S-drone, a fully open-source UAV platform and simulation stack that supports onboard

sensing and cooperative inter-robot tracking, enabling demonstrations of swarm behaviors (e.g., formation flight, cooperative transport, wide-area coverage) without external infrastructure. The second is the Self-organizing Nervous System (SoNS), which contributes self-organizing hierarchy for robot swarms. Using the SoNS, robots self-organize into temporary locally centralized networks that fuse information and coordinate actions; these networks can split or merge and any robot acting as a temporary leader can be replaced on the fly, allowing the swarm to reconfigure as conditions change, while maintaining scalability and fault tolerance against the loss of arbitrary robots. The third is Hierarchical Henneberg Construction (HHC), which contributes a graph-theoretic formalization of the SoNS for aerial swarms, based on bearing (angle of arrival). The HHC algorithms yield analyzable procedures to construct, split, merge, and rebuild self-organized frameworks while preserving rigidity and hierarchy. Finally, the fourth contribution is a proactive–reactive fault-tolerance strategy for intermittent faults in aerial swarms, based on the introduction of Adaptive Biased Minimum Consensus (ABMC). The ABMC is a distributed protocol that constructs and maintains low-cost backup paths to leaders in self-organized hierarchical swarms. The paths constructed using ABMC support one-shot likelihood-ratio tests to detect faulty information originating in a portion of the swarm, then supply the backup paths that enable quick rerouting to sustain information flow and stable formation control. Together, these contributions provide an open experimental platform, a reconfigurable hierarchical control paradigm, and provable resilience tools—bridging the gap between theory and practice in resilient coordination in aerial robot swarms.

Acknowledgements

All praise is due to Allah, the Most Gracious, the Most Merciful.

I would like to express my sincere gratitude to my supervisors, Marco DORIGO, Emanuele GARONE and Mary Katherine HEINRICH. It has been a privilege to work under their mentorship, and I am profoundly grateful for the opportunity they gave me, as well as for their invaluable advice and constant support throughout this journey.

I owe a very special debt of appreciation to Mary Katherine HEINRICH. Her dedication, encouragement, and timely guidance were instrumental in completing this thesis; without her, this work would not have been possible.

I also extend my thanks to the members of my thesis committee for their constructive feedback, careful evaluation, and for devoting their time to examine my thesis.

I am grateful to all my colleagues at IRIDIA and SAAS, whose companionship and collaboration made this journey enriching. In particular, I wish to thank Weixu ZHU and Michael ALLWRIGHT for their consistent technical support and thoughtful input.

My heartfelt thanks go to Muriel DECRETON and Pascale LATHOUWERS, whose efficiency, kindness, and availability ensured that everything ran smoothly. I also appreciate the invaluable assistance of Laurent CATOIRE and Serge TORFS, for their expertise and practical support.

I remain indebted to my friends in Türkiye, whose encouragement has always been a source of strength, especially Muhammed Emin ALDEMİR, Mustafa Murat SEZER, İbrahim Tayyip İŞLER, Abdurrahman BAYRAK and Mehmet Altan

TOKSÖZ.

To my parents, I offer my deepest gratitude for their love, sacrifices, and prayers that have carried me forward.

Finally, I wish to acknowledge my beloved wife, Beyza, whose unwavering support, patience, and love have been my greatest source of strength. This accomplishment is as much hers as it is mine.

Contents

Chapter 1

Introduction

Unmanned aerial vehicles (UAVs)—commonly known as drones—have advanced the field of robotics by offering highly versatile mobility. Their capability to navigate three-dimensional aerial space enables them to access difficult-to-reach areas and perform tasks in dynamic, unstructured environments, significantly expanding the potential applications of robotic technology. A major contributor to the growth and accessibility of modern UAVs is the substantial reduction in development and deployment costs, largely driven by the proliferation of open-source hardware and software platforms (Aliane, 2024). These platforms facilitate innovation and customization through collaborative developer communities, creating a vibrant ecosystem for researchers, hobbyists, businesses, and government agencies to construct sophisticated drones without incurring prohibitive costs. Additionally, advancements in microelectronics have led to the creation of smaller, more power-efficient components. As a result, UAVs have become increasingly compact, affordable, and broadly accessible to diverse users across civil and defense sectors (Doornbos et al., 2024).

Thanks to these enabling factors, UAVs are now deployed in an expanding range of civil applications. Recent surveys highlight drone use in environmental monitoring and precision agriculture, smart city infrastructure inspection, disaster management and search-and-rescue operations, traffic surveillance, delivery of goods, and more (Ariante and Del Core, 2025). In many of these domains, drones offer practical advantages because they are easy to deploy, low-maintenance, and highly mobile. For example, in emergency response scenarios UAVs can be rapidly launched for

real-time damage assessment or to deliver medical supplies, tasks that benefit from drones' low operational costs and ability to reach hazardous areas where manned access is risky (Tahir et al., 2023).

The widespread adoption of UAVs across diverse industrial sectors has exposed critical operational challenges that demand advancements in vehicle autonomy, robustness, and operational efficiency, while simultaneously requiring the integration of multimodal sensing technologies to enable enhanced situational awareness and adaptive decision-making in complex environments. Despite their cross-domain adaptability, even minor modifications to UAV design or operational objectives can introduce crucial inefficiencies. To illustrate this tension between versatility and optimization, consider a lightweight UAV platform originally designed for agricultural monitoring using multispectral cameras. Expanding its operational scope to include LiDAR-based topographical mapping would impose conflicting requirements: the LiDAR system's additional weight and power demands could exceed the UAV's payload capacity and battery endurance. These interrelated limitations—restricted payload capacity, finite flight endurance, and constrained sensing range—fundamentally restrict standalone UAVs in addressing large-scale or complex missions (Watts et al., 2012). Such constraints have catalyzed research into collaborative multi-agent systems, in which UAV teams share sensing, computational, and actuation resources to overcome individual platform constraints (Rizk et al., 2019).

Recent research into multi-UAV systems has included swarm architectures to make coordinated teams of aerial robots capable of decentralized and cooperative operation (Chung et al., 2018; Skorobogatov et al., 2020). Often drawing inspiration from principles of collective behaviors observed in nature, such as bird flocking and insect colony coordination, swarm robotics seeks to engineer scalable and adaptive group dynamics through local interactions (Brambilla et al., 2013; Şahin, 2004). However, translating these principles into functional UAV swarms introduces unique challenges. Unlike the simple ground-based robotic platforms predominantly used in swarm robotics research—which enable controlled experimentation in simplified 2D environments with minimal risk of hardware damage or safety hazards (Dorigo et al., 2020, 2021)—UAV swarms must operate in 3D aerial environments with complex dy-

namics, limited onboard processing, and stringent safety requirements (Chung et al., 2018). Unlike the inherent safety of laboratory-scale ground platforms that allow researchers to iterate algorithms freely, treating collisions or failures as inconsequential learning opportunities, in UAV swarms even minor miscalculations—such as trajectory conflicts or sensor misalignments—can lead to catastrophic collisions or mission failures (Schiano et al., 2016).

These operational constraints in UAV swarms underscore a critical dependency in robot swarms in general: the performance of a robot swarm in any mission depends fundamentally on the suitability of its control strategy for the given task. It is well known that fully centralized control of large-scale multi-robot systems poses several problems, including limited scalability, a single point of failure in the coordinating agent, and potentially unrealistic communication infrastructures. To circumvent these problems, the swarm robotics community has demonstrated that a group of robots can be controlled in a completely decentralized way (Dorigo et al., 2004, 2013; Li et al., 2019; Nouyan et al., 2009; Rubenstein et al., 2014; Werfel et al., 2014). However, as the size and speed of a fully decentralized swarm increases, the design and management of swarm behaviors becomes increasingly difficult. For example, maintaining formations in UAV swarms requires precise relative state estimation and consensus protocols (Schiano et al., 2016), yet these processes are inherently error-prone (especially so in UAVs) due to sensor noise, communication delays, or occlusion, and decentralized behaviors can become impractical (e.g., rapid state transitions outpace the convergence of local peer-to-peer interactions between robots.) Without a central coordinator, UAVs cannot directly receive information about the swarm's global state, forcing them to make decisions based on fragmented local observations. Such issues highlight the paradox of fully decentralized swarms: while local interactions enable scalability, flexibility, and fault tolerance—as often demonstrated in controlled laboratory settings with low-speed ground robots (Bjerknes and Winfield, 2013; Dorigo, 2005; Rubenstein et al., 2014; Tarapore et al., 2017; Valentini et al., 2016)—real-world operational requirements such as observability (the ability to monitor swarm states), manageability (the capacity to direct or adjust behavior), and robustness (resilience to failures or disturbances), alongside dynamic environ-

ments and hardware imperfections, demand new strategies to harmonize individual autonomy with swarm-wide coordination.

One critical vulnerability in UAV swarms that urgently demands new strategies is their fault tolerance capabilities, particularly under dynamic real-world conditions. Although swarm robotics typically studies robustness against full robot shutdowns, empirical studies have meanwhile demonstrated that even minor faults in an otherwise operational individual robot can significantly degrade collective performance (Winfield and Nembrini, 2006). The challenge of addressing this vulnerability intensifies with intermittent faults (IFs), which exhibit stochastic spatio-temporal characteristics that complicate detection and mitigation (Zhou et al., 2019). Left unmitigated, such faults risk triggering cascading failures (Niu et al., 2021) that destabilize overall swarm coordination. A representative example involves intermittent GPS signal degradation in cluttered environments, which can induce sporadic localization errors. These errors propagate through decentralized state estimation protocols, gradually undermining coordination mechanisms without generating explicit failure indicators.

Existing fault-tolerance strategies in robot swarms predominantly adopt a reactive paradigm and assume faults can be resolved post-occurrence, either via autonomous recovery or human intervention (Christensen et al., 2009). This assumption, however, fails in scenarios where temporal constraints render reactive approaches inadequate, for example in inaccessible environments, hazardous zones, or enclosed spaces prone to rapid congestion (Dorigo et al., 2020, 2021). Delayed fault mitigation in such cases exacerbates instability. These limitations underscore the necessity for proactive fault tolerance strategies to achieve robust long-term autonomy (Bjerknes and Winfield, 2013). By anticipating and mitigating faults during their latent phase—that is, before they manifest as operational failures—robot swarms could minimize potential performance degradation.

In summary, the following interconnected research challenges are central to advancing reliability and robustness in aerial robot swarms:

1. **Coordination:** Fully decentralized swarm architectures often lack analytical

tractability, making it difficult to ensure coordinated actions in dynamic environments (Brambilla et al., 2013). The absence of formal guarantees limits predictability in swarm coordination and complicates synchronization, task allocation, and communication protocols among robots (Hamann, 2018). Moreover, designing reliable coordination strategies often depends on empirical testing or heuristic approaches, which might fail to transfer effectively to diverse mission scenarios, larger swarm sizes, or different robot types (Dorigo et al., 2020, 2021).

2. **Fault tolerance:** Although robot swarms feature some inherent fault tolerance through redundancy, especially to permanent faults, empirical evidence has shown that partial or intermittent faults in a subset of robots can disproportionately degrade swarm performance (Bjerknes and Winfield, 2013; Winfield and Nembrini, 2006). Conventional fault-detection techniques frequently assume reliable communication channels or self-diagnostic capabilities within individual robots (O'Keeffe and Millard, 2023). However, realistic operating conditions often involve degraded communication links, hardware malfunctions, and noisy sensor data, which can delay fault detection and result in ineffective recovery strategies (Dorigo et al., 2020, 2021).

To address these challenges, this thesis presents a comprehensive framework that spans hardware development, theoretical analysis, and practical implementation, to build reliable and fault-tolerant aerial robot swarms.

It begins by introducing the S-drone (Oğuz et al., 2024), an open-source UAV platform tailored for swarm robotics research. The platform combines onboard sensing with cooperative inter-robot tracking, providing a versatile and practical testbed for swarm robotics approaches. Specifically, the S-drone platform is pre-configured to detect fiducial markers in the environment, leveraging this sensor information to track the presence, identity, relative position, and relative orientation of neighboring peers using cooperative feature-level sensor fusion for inter-robot tracking. Additionally, the S-drone has open-access hardware and open-source software, an open-source simulation environment for development, and an open-source user inter-

face for operators to manage demonstrations with real UAVs. The S-drone approach for cooperative feature-level sensor fusion is adaptable, allowing integration with other UAV fleets if the necessary payloads and platform customization options are available. By bridging the gap between theory and practice, the S-drone platform lays the groundwork for experimental validation of large-scale aerial swarms.

Building upon the S-drone platform, the thesis presents a novel approach to achieving resilient coordination in robot swarms, demonstrated in a heterogeneous swarm of S-drones and ground robots: the Self-organizing Nervous System (SoNS) (Zhu et al., 2024). The SoNS enables robots to temporarily assume "brain" (leader) roles over other robots, coordinating sensing, actuation, and decision-making within temporarily centralized sub-swarms. In practice, this means that while each robot operates autonomously, they self-organize into hierarchical sub-swarms with dynamic leaders. These leaders use the self-organizing hierarchical structure to aggregate local information and guide the collective behavior of the robots, thereby creating flexible, temporarily centralized structures that enhance the overall responsiveness and robustness of the swarm. The swarm can quickly adapt to dynamic environments using centralized coordination when needed, but still retain interchangeability of robots typical in self-organized systems (i.e., any robot can become the brain, and can replace a former brain if it fails).

The proof-of-concept experiments with ground and aerial robots enabled by the SoNS in (Zhu et al., 2024) show great practical benefit to robot swarms. However, extending this hierarchical paradigm to large-scale homogeneous UAV swarms would require both robust theoretical frameworks for dynamically restructuring communication links and advanced fault-tolerance mechanisms that can handle real-world uncertainties. Unlike the slower ground robots included in (Zhu et al., 2024), homogeneous UAV swarms can operate in three-dimensional, highly dynamic environments where rapid changes in position and connectivity are the norm. This dynamism means that the swarm must be able to reconfigure quickly and accurately to navigate obstacles or respond to sudden robot malfunctions. For instance, a formation might need to split into multiple sub-formations to maneuver through a narrow passage and then merge seamlessly to continue its mission, or reconfigure instantly to

maintain stability after several units fail unexpectedly due to a collision. To address these challenges, this thesis proposes mathematical methods for systematically building, merging, splitting, and reorganizing self-organized hierarchical communication frameworks. These methods ensure that the swarm can dynamically adapt its structure while preserving connectivity and stability, enabling it to maintain coordinated behavior as roles and inter-robot links evolve.

In addition to flexible communication frameworks, SoNS must incorporate advanced fault-tolerance mechanisms to handle real-world uncertainties. Using local subgroup coordination, the SoNS already mitigates interference risks and avoids uncertainties arising from misinterpreted roles or conflicting priorities—pitfalls that can undermine flat swarm systems. It also mitigates risks from permanent failures (full shutdown) of any individual robots, including the brain. However, the SoNS alone does not address how to detect and recover from faults in transmitted information. Tolerance to informational faults is especially important when the faults are transient, as intermittent faults can be difficult to distinguish from normal behavior and can destabilize the hierarchy and the coordinated swarm behaviors. Intermittent faults can be caused by sensor inaccuracies, hardware malfunctions, unpredictable environmental conditions, or communication disruptions. To address this gap in SoNS fault tolerance, the thesis introduces a proactive–reactive fault-tolerance strategy that extends the SoNS paradigm. Proactively, robots establish backup self-organized communication paths to the brain (leader) robot and use these paths to detect the presence of intermittent faults. Reactively, when such faults arise, the swarm then uses the proactively constructed backup paths to reroute information flows and isolate the faulty links. This approach ensures that momentary glitches do not escalate, and transient but persistent inaccuracies are isolated from the rest of the swarm for their duration, preserving overall functionality even under uncertainty. In short, by leveraging the temporarily centralized monitoring available via the SoNS network, individual robots can cross-check their observations and apply corrective measures before faults propagate throughout the swarm.

1.1 Thesis structure and related scientific contributions

This section describes the structure of the thesis and lists the related scientific papers published in international conferences and journals.

The dagger symbol (†) signifies that the authors contributed equally and are co-first authors of the published article.

Chapter 2 surveys the state of the art in aerial swarm robotics, including hardware platforms, formation control, coordination and communication architectures, and fault-tolerance strategies, and identifies critical gaps that motivate the proposed framework. The chapter is partially based on the related work sections from the following peer-reviewed articles:

- **Oğuz, S.**, Heinrich, M.K., Allwright, M., Zhu, W., Wahby, M., Garone, E. and Dorigo, M., 2024. An Open-Source UAV Platform for Swarm Robotics Research: Using Cooperative Sensor Fusion for Inter-Robot Tracking. *IEEE Access*, 12, 43378-43395.

- Zhu, W.†, **Oğuz, S.**†, Heinrich, M.K.†, Allwright, M., Wahby, M., Christensen, A.L., Garone, E. and Dorigo, M., 2024. Self-organizing nervous systems for robot swarms. *Science Robotics*, 9(96), eadl5161.

- Zhang, Y., **Oğuz, S.**, Wang, S., Garone, E., Wang, X., Dorigo, M. and Heinrich, M.K., 2023. Self-reconfigurable hierarchical frameworks for formation control of robot swarms. *IEEE Transactions on Cybernetics*, 54(1), 87-100.

- **Oğuz, S.**, Garone, E., Dorigo, M., and Heinrich, M.K. (2025). Proactive–reactive detection and mitigation of intermittent faults in robot swarms. Submitted to *IEEE Transactions on Robotics*. Pre-print available at arXiv:2509.19246.

Chapter 3 introduces the S-drone, an open-source UAV platform designed for swarm robotics research. Equipped with onboard sensing and local interaction capabilities, the S-drone enables rigorous testing and evaluation of swarm robotics algorithms in real-world scenarios. This chapter details the platform's hardware design, software components, and the accompanying simulation environment. The

same platform serves as the foundation for the experiments presented in Chapter 4 of this thesis. This chapter is based on the following peer-reviewed article published in an international journal (with the preliminary results having been presented at a regional conference, also listed below):

- **Oğuz, S.**, Heinrich, M.K., Allwright, M., Zhu, W., Wahby, M., Garone, E. and Dorigo, M., 2024. An Open-Source UAV Platform for Swarm Robotics Research: Using Cooperative Sensor Fusion for Inter-Robot Tracking. *IEEE Access*, 12, 43378-43395.

- **Oğuz, S.**, Heinrich, M.K., Allwright, M., Zhu, W., Wahby, M., Dorigo, M., and Garone, E. (2022). A novel aerial robotic swarm hardware. In *Proceedings of the 41st Benelux Meeting on Systems and Control: Book of Abstracts* (p. 96). Brussels, Belgium.

Chapter 4 introduces a self-organizing hierarchical coordination architecture called the Self-organized Nervous System (SoNS), which allows robots to dynamically assume or relinquish leadership roles based on local context. The chapter illustrates how hierarchy is self-organized and demonstrates the practical implementation of SoNS in a heterogeneous swarm of S-drones and ground robots. This chapter is based on the following peer-reviewed article published in an international journal:

- Zhu, W.†, **Oğuz, S.**†, Heinrich, M.K.†, Allwright, M., Wahby, M., Christensen, A.L., Garone, E. and Dorigo, M., 2024. Self-organizing nervous systems for robot swarms. *Science Robotics*, 9(96), eadl5161.

The contributions I made to this work are as follows. I developed a mathematical formulation of the SoNS framework and its theoretical guarantees, including the stability criteria and proofs of convergence. On the implementation side, I contributed to the formalization of the Allocate algorithm (Section 4.2.2), which enables each robot to autonomously assign and reassign its children at runtime by comparing current and target graph states and thereby ensures reconfiguration under faults, as well as the Classify routine (Section 4.2.2), which categorizes sensory inputs into

local reactions, SoNS-wide responses, or upstream hand-offs, and is integrated with the unified BrainProgram (Section 4.2.3). Finally, I co-conducted the experiments, the data collection, and the presentation of the experimental results.

Chapter 5 presents algorithms designed to allow robots to dynamically modify who communicates with whom, thus preserving network connectivity during formation control in aerial robot swarms. The chapter proposes a theoretical framework grounded in bearing rigidity and Henneberg-type construction, particularly for self-organized hierarchical networks of robots applied to formation control tasks. This chapter introduces mathematically provable, analyzable algorithms for hierarchical frameworks, providing the foundations for merging, splitting, and reconstructing robot formations to preserve connectivity and stability under evolving mission requirements or robot failures. This chapter is based on the following peer-reviewed article published in an international journal:

- Zhang, Y., **Oğuz, S.**, Wang, S., Garone, E., Wang, X., Dorigo, M. and Heinrich, M.K., 2023. Self-reconfigurable hierarchical frameworks for formation control of robot swarms. *IEEE Transactions on Cybernetics*, 54(1), 87-100.

The contributions I made to this work are as follows. I assisted in the literature review and gap analysis. I performed preliminary simulations to establish the foundation for formulating the proposed approach, contributing to the subsequent theoretical and experimental investigations. I contributed to the theoretical development by formalizing and verifying key lemmas on bearing persistence in directed, lower-triangular graphs. I contributed to the experimental design by defining the four formation-control scenarios and specifying the performance metrics.

Chapter 6 addresses intermittent faults (IFs): transient faults caused by sensor glitches, communication interference, or software bugs that may appear, vanish, and reappear unpredictably. Such IFs can propagate corrupted information before detection. To counter this, consensus- and statistical-estimation–based methods are integrated with a proactive strategy: before faults occur, robots preemptively establish backup communication links ensuring the swarm is prepared for disruptions. When an IF emerges, these enhanced links enable rapid fault isolation, preventing

cascading failures and preserving overall system reliability. The results presented in this chapter are currently under review for publication in an international scientific journal:

- **Oğuz, S.**, Garone, E., Dorigo, M., and Heinrich, M.K. (2025). Proactive–reactive detection and mitigation of intermittent faults in robot swarms. Submitted to *IEEE Transactions on Robotics*. Pre-print available at arXiv:2509.19246.

Finally, Chapter 7 summarizes the contributions of this thesis, discusses their implications for the future of aerial swarm robotics, and highlights promising directions for future research.

Chapter 2

Related Work

Swarm robotics research has seen significant advancements over recent decades, particularly in deploying real-world swarms of small, mobile ground robots in laboratory environments (Dorigo et al., 2020). While aerial swarms have also achieved notable milestones with practical demonstrations of swarm algorithms on small scale UAV platforms (e.g., quadrotors (Baca et al., 2021; Dorigo et al., 2013; Zhou et al., 2022)), these efforts remain exceptions (Skorobogatov et al., 2020). As a result, much of the work in aerial swarm coordination still relies on simulation, with only limited validation on physical UAV platforms (Coppola et al., 2020).

Moving from simulated environments to real-world testing hinges on one of swarm robotics' fundamental principles: decentralized, local interactions. In a true swarm, each agent makes decisions using only information from its immediate neighbors, rather than following commands from a central controller (Şahin, 2004). This local decision-making is what grants swarms their celebrated scalability, flexibility, and some degree of inherent fault tolerance. However, extending this principle to aerial vehicles introduces new layers of complexity. Unlike their ground-based counterparts, UAVs navigate in three dimensions at high speeds and must cope with aerodynamic effects, wind disturbances, and dynamic obstacles (Shakhatreh et al., 2019). Achieving the real-time detection, tracking, and peer-to-peer communication of positions and orientations that high-speed flight demands remains an ongoing challenge (Chung et al., 2018), and is a key reason many published algorithms have not yet left simulation (Abdelkader et al., 2021).

Moreover, these real-time coordination hurdles are compounded by practical hardware constraints. Most commercial UAVs are optimized for solo missions and simply lack the onboard sensing suites, processing power, and robust communication links required for fully decentralized operation (Campion et al., 2018; Skorobogatov et al., 2020). While specialized research platforms provide tailored solutions for individual capabilities—such as obstacle avoidance or high-speed flight—they frequently fail to satisfy the comprehensive operational requirements of swarm robotics: the hardware and software needed to support a robot swarm with scalability, robustness, decentralization, adaptability, and general-purpose versatility (Bayindir and Şahin, 2007; Brambilla et al., 2013; Dias et al., 2021; Dorigo et al., 2020, 2021; Heinrich et al., 2019). Consequently, despite the theoretical extensibility of many swarm algorithms to aerial platforms, large-scale experimental validation in real flight scenarios remains scarce (Abdelkader et al., 2021; Coppola et al., 2020).

Effective aerial swarm coordination demands sophisticated control strategies. While centralized control may be simpler to implement, it creates single points of failure and struggles to scale with swarm size; in contrast, fully decentralized control, although more resilient, can lead to unpredictable behavior in dynamic settings where quick decisions are critical (Dorigo et al., 2021). To address this, hierarchical control architectures are a promising middle ground (Dorigo et al., 2020; Jamshidpey et al., 2020, 2023, 2025; Ju and Son, 2021). Using such an architecture, temporary leadership roles have the potential to be dynamically assigned within the swarm, balancing autonomy with structured coordination. Such approaches could enhance decision-making efficiency and resilience, but these benefits must be supported by analytical tools to ensure system stability as well as strategies to maintain operational continuity under uncertainty.

In real-world scenarios, aerial swarms also face the need to maintain precise formations while navigating complex, obstacle-rich environments (Zhou et al., 2022). Whether using leader-follower schemes, virtual structures, or decentralized consensus protocols, ensuring collision avoidance and shape preservation using local measurements is notoriously difficult without a global reference (Liang et al., 2016; Oh et al., 2015; Zhang and Liu, 2019; Zhou et al., 2018). To address this challenge, a popular

approach is to use rigidity theory – a mathematical framework originally from graph theory and mechanics – as the foundation for maintaining a swarm's shape (Montijano et al., 2016; Zhao and Zelazo, 2019). Rigidity theory, particularly bearing (i.e., angle of arrival) rigidity, offers a useful mathematical basis for formation stability using strictly local onboard measurements (Michieletto et al., 2016). Adapting this theoretical framework to real-world scenarios can enable aerial swarms to preserve their structure without relying on a global reference, allowing them to respond to task changes or unexpected disruptions in a self-organized manner.

Another important challenge for aerial swarms is communication reliability: many swarm models assume ideal, bidirectional links, but real environments often introduce intermittent, asymmetric connectivity due to obstacles, interference, and variable signal strength (Wang et al., 2024). This gap between theoretical models and practical implementation is a fundamental challenge for aerial swarm robotics, as the robustness of an aerial swarm often critically depends on continuous and accurate information exchange (Tarapore et al., 2020). Because of these inherent communication limitations, no aerial swarm can be truly robust if it lacks effective strategies for dealing with failures or unanticipated anomalies. Even if a swarm is designed to include interchangeable robots and use strictly decentralized decision-making, unexpected transient errors can severely undermine overall performance (O'Keeffe, 2025).

Intermittent faults (IFs)—temporary, sporadic faults such as brief sensor malfunctions or transient communication errors—are particularly concerning because they can go unnoticed at first yet later propagate through the swarm, causing cascading failures (Peng et al., 2021). In aerial swarms, where rapid and precise coordination is essential, these transient errors can quickly propagate and exacerbate, compromising the entire mission. For instance, a brief communication or sensor glitch in a few robots might lead to incorrect state estimation in several more robots, culminating in misaligned movements or even collisions within the swarm (Hu et al., 2022). Existing fault-tolerance strategies in swarm robotics have usually focused on passive tolerance or post-failure repair, addressing issues only after they have noticeably impacted behavior (Peng et al., 2021), which might be too late for many types of IFs, especially in aerial operations. The unpredictable nature of IFs in aerial swarms

demands a more proactive strategy—one that can detect early signs of failure and isolate or correct faults before they have the chance to spread through the swarm.

Taken together, these themes—algorithmic validation, hardware adequacy, control architecture, formation stability, and fault tolerance—constitute the current landscape of challenges in aerial swarm robotics. This chapter reviews the existing research relevant to these challenges, organized according to the topics covered in Chapters 3, 4, 5, and 6. It begins with an examination of current UAV platforms, their capabilities, and the gaps in their application to swarm robotics research. The limitations inherent in commercially available systems are discussed alongside promising advancements in sensing and inter-robot localization that pave the way for improved aerial swarm deployment. This chapter then discusses the foundations of robot swarm control, in particular the trade-offs present when selecting either decentralization or centralization, thus motivating hybrid approaches such as self-organizing hierarchy to enhance coordination and manageability. An overview of formation control strategies for aerial swarms is then provided, followed by a discussion of rigidity theory—a mathematical framework used to ensure stability and connectivity in multi-robot formations. Although the literature presents methods for framework construction and formation maintenance via rigidity theory, the systematic and mathematically analyzable organization and reorganization of hierarchical frameworks remains an open problem. Finally, the chapter reviews fault-tolerance strategies in swarm robotics, focusing on the challenges associated with addressing intermittent faults (IFs). Although a robot swarm will often exhibit inherent fault tolerance against the permanent failure of some of its members, current approaches do not address the detection or mitigation of IFs in a swarm, which are transient and unpredictable but can spread quickly and potentially cause permanent damage.

2.1 Hardware platforms for aerial swarm robotics

There are currently many UAV platforms available off-the-shelf, but the majority of these platforms do not have open hardware and software and therefore are not ideal for scientific research. The exceptions are commercial open development platforms

and open scientific research platforms, of which the most relevant for swarm robotics are: (1) the ModalAI M500 drone[1] for GPS-denied navigation and obstacle avoidance using stereo cameras and (2) the *Palm-Sized Drone* (Zhou et al., 2022) for swarm navigation in cluttered outdoor environments using ultra-wideband (UWB) distance-based localization drift correction. While both of these platforms offer advanced capabilities for swarm navigation, neither the ModalAI M500 platform nor the Palm-Sized Drone platform come pre-equipped with the ability to locally identify and track the full relative poses (i.e., positions and orientations) of nearby drones. It might be feasible to extend the software of either platform to achieve these capabilities in their targeted navigation contexts using their current off-the-shelf payloads, but this would of course be a non-trivial development task.

Based on available information, no commercial drone (whether open or fully proprietary) comes pre-equipped with the ability to identify and track the poses of nearby drones using only onboard sensors and computation. However, research on sensing technologies is advancing quickly, for example using UWB localization (Fishberg and How, 2022; Morón et al., 2022; Xu et al., 2022; Zhou et al., 2022) or visual localization supported by machine learning (Chen et al., 2023; Valada et al., 2018; Walter et al., 2020). UWB state estimation is very promising, and advances are being made to address outstanding challenges, such as insufficient accuracy, complicated initialization, and consistency issues in UWB-odometry state estimation (Xu et al., 2022) as well as scalability issues related to communication throughput in UWB-odometry or UWB-visual-inertial state estimation, especially for 2D pose estimation (Fishberg and How, 2022). Another very promising technology is ultraviolet direction and ranging (UVDAR) (Ahmad et al., 2021; Walter et al., 2019). UVDAR can allow drones to identify and track the positions and orientations of nearby drones using only onboard sensors and computation, and is currently being implemented as an extension (Hert et al., 2023) to the open-source MRS research platform (Baca et al., 2021). In the mid-term future, it is likely that some of these technologies will become available on commercial drone platforms that are open and come pre-built off-the-shelf.

[1] https://docs.modalai.com/m500/

Given the current limitations in commercially available UAV platforms and the rapid advancements in sensing technologies, it is useful to examine existing research-focused UAV platforms to evaluate their suitability for swarm robotics experiments. Table 2.1 provides a comparative summary of their characteristics, highlighting key features such as hardware design, sensing capabilities, and software openness.

The FLA platform (Mohta et al., 2018), designed for obstacle detection and environmental mapping, is based on the DJI F450 with a 450 mm wheelbase (motor to motor distance). It incorporates a Hokuyo 2D LiDAR (light detection and ranging), a downward-facing Garmin LiDAR-Lite rangefinder, multiple cameras, a high-performance VectorNav VN-100 inertial measurement unit (IMU), and is powered by an Intel NUC i7 onboard computer. It weighs 3 kg, has a flight duration of 5 minutes, and is estimated to cost approximately 5 500 EUR. However, its software and details of its hardware design are undisclosed.

The ASL-Flight platform (Sa et al., 2017), designed for verification of custom visual inertial odometry framework, is based on the proprietary DJI Matrice 100 with a 650 mm wheelbase. It is equipped with an Intel RealSense ZR300 depth imaging sensor and an Intel NUC i7 onboard computer. Weighing 2.4 kg, it has a flight duration of 16 minutes and is estimated to cost approximately 4 700 EUR. The UAV's hardware is proprietary to DJI and the details of its design are undisclosed. Its VIO (visual inertial odometry) framework is also not open-source.

The Agilicious platform (Foehn et al., 2022), designed for autonomous agile flight, utilizes an Armattan Chameleon frame with a 264 mm wheelbase. It incorporates an Intel RealSense T265 tracking camera and an NVIDIA Jetson TX22 onboard computer. It has a flight duration of 10 minutes, weighs 750 grams and has an estimated cost of approximately 1 580 EUR. Even though it is open-source, there are some access restrictions based on criteria such as the requester's nationality.

Table 2.1: Characteristics of existing UAV research platforms (i.e., non-commercial)

Platform	**FLA(Mohta et al., 2018)**	**ASL-Flight(Sa et al., 2017)**	**Agilicious(Foehn et al., 2022)**	**MRS(Baca et al., 2021)**	**Palm-Sized Drone(Zhou et al., 2022)**
Target application	Obstacle detection, environment mapping	Visual-inertial odometry	Autonomous agile flight	Underground search and rescue	Multi-UAV navigation
Base model	DJI F450	DJI Matrice 100	Armattan Chameleon	Holybro X500	Not specified
Wheelbase (mm)	450	650	264	500	114
Sensors (Distance/Depth)	Hokuyo 2D Lidar; Garmin rangefinder	Intel RealSense ZR300	None specified	Ouster OS1 Lidar; Intel RealSense D435i	Intel RealSense D430
Camera	Multiple	None specified	Intel RealSense T265	x2 Basler RGB cameras	Grayscale camera
IMU (Internal Measurement Unit)	VectorNav VN-100	None specified	None specified	None specified	None specified
Onboard computer	Intel NUC i7 mini-PC	Intel NUC i7 mini-PC	NVIDIA Jetson TX22	Intel NUC i7 mini-PC	NVIDIA Xavier NX
Weight (g)	3000	2400	750	4800	300
Flight duration (min)	5	16	10	25	11
Cost (EUR, approx.)	5500	4700	1580	12000	1450
Software/ Hardware design	Undisclosed	Hardware proprietary to DJI	Available on request	Hardware undisclosed; open-source	Open-source
Local pose tracking of nearby UAVs	No	No	No	In development (extension)	No

The MRS platform (Baca et al., 2021), designed for underground search and rescue operations, utilizes a Holybro X500 frame which has a 500 mm wheelbase. It is equipped with an Ouster OS1 LiDAR, one Intel RealSense D435i depth camera, two Basler RGB cameras and an Intel NUC i7 onboard computer. Weighing 4.8 kg, it has a flight duration of 25 minutes and is estimated to cost 12 000 EUR. The software is open-source and system-level design is partially open-source. However, the details of its hardware design are undisclosed.

The Palm-Sized Drone (Zhou et al., 2022), designed for multi-UAV navigation in cluttered outdoor environments, has a 114 mm wheelbase. It carries an Intel RealSense D430 depth camera and a grayscale camera, and it is powered by an NVIDIA Xavier NX onboard computer. It has a flight duration of 11 minutes, weighs 300 grams, and is estimated to cost approximately 1 450 EUR. Both its software and hardware designs are open-source.

Besides these research platforms another very well-known platform is the Crazyflie (Giernacki et al., 2017), which has an 80 mm wheelbase, weighs 27 grams and is estimated to cost 220 EUR. However, due to its minimal hardware configuration, it is primarily used for basic algorithm verification tasks, as it does not support onboard sensing or computation capabilities, and therefore this chapter does not report it in Table 2.1.

In conclusion, while existing UAV platforms offer various capabilities for navigation, mapping, and agile flight, they often lack integrated support for local inter-UAV pose tracking using only onboard sensors and computation.

2.2 Hierarchical swarm control and coordination

Over the past two decades, the swarm-robotics community has established that large populations of autonomous agents can coordinate effectively in the absence of any central coordinating entity, thereby achieving the canonical advantages of scalability, flexibility, and robustness to failures of individual robots. Collective behaviors have been realized for environmental monitoring, distributed navigation and payload transport, programmable self-assembly, construction, and bio-hybrid interac-

tion (Dorigo et al., 2013; Halloy et al., 2007; Petersen et al., 2019; Rubenstein et al., 2014; Talamali et al., 2021; Wahby et al., 2018; Werfel et al., 2014). Notwithstanding these experimental successes, the transition of flat (single-level) swarms from laboratory prototypes to real-world deployments remains limited (Dorigo et al., 2020). Because macroscopic objectives must emerge from local interactions, the synthesis of suitable low-level control policies is analytically intractable for many tasks (Hamann, 2018); development therefore relies heavily on heuristic tuning, extensive testing in simulation, and empirical iteration (Hamann, 2010; Hamann and Wörn, 2008). Moreover, convergence times may be prohibitive when rapid response is essential, and performance often degrades when environmental conditions deviate from those foreseen during design.

Conversely, strictly centralized architectures offer well-developed algorithmic toolchains for a wide array of complicated tasks, such as simultaneous localization and mapping (Howard, 2006), dynamic task allocation, and online vehicle routing (Psaraftis et al., 2016). Centralization simplifies behavior design and facilitates the integration of heterogeneous capabilities, yet full centralization inevitably introduces bottlenecks and single points of failure that impair scalability and resilience—properties ordinarily regarded as the *raison d'être* of robot swarms.

In current multi-robot platforms, the type of communication topology and control distribution is typically fixed *a priori*. For example, in canonical robot swarms, a strict heterarchy is adopted in which agents communicate peer-to-peer without ranks. Such flat designs, exemplified by optimized aerial flocking (Vásárhelyi et al., 2018), align well with the field's biological inspiration sources (Beni, 1988; Bonabeau et al., 1999; Floreano and Mattiussi, 2008; Hamann, 2018; Şahin, 2004) and offer some scalability, flexibility, and fault-tolerance benefits. Nevertheless, a statically flat interaction structure can complicate global behavior design and curtail transferability to unforeseen operational demands.

Biological collectives demonstrate that strict heterarchy is not the only option for self-organized systems. Ant colonies can employ division of labor; bird flocks and fish schools can exploit transient leadership by informed individuals (Groß et al., 2008; Varadharajan et al., 2024). Several research strands in swarm robotics have

begun to explore this design space. Behavioral heterogeneity has been used to induce implicit leadership whereby better-informed, more persistent, or human-supervised agents influence group decisions (Balázs et al., 2020; Firat et al., 2020; Valentini et al., 2016; Walker et al., 2014a,b). Double-level systems with single broadcast leaders have demonstrated reliable information dissemination (Kaiser and Hamann, 2022; Shan and Mostaghim, 2020), and numerous leader–follower strategies for flocking and self-assembly have been reported (Amraii et al., 2014; Dalmao and Mordecki, 2011; Gu and Wang, 2009; Jia and Vicsek, 2019; Pignotti and Vallejo, 2018; Zheng et al., 2020). Multi-drone planners that combine centralized optimization with decentralized collision avoidance have achieved high performance in dense environments (Zhou et al., 2022). However, a common limitation of these approaches is that the leadership roles and inter-agent dependencies are either predefined or remain static once assigned; few mechanisms exist for autonomously reconfiguring the hierarchy in response to internal failures or external perturbations.

Consequently, addressing the open problem of a *self-organized, controllable hierarchy* is not only pivotal for swarm robotics in general but arguably becomes indispensable when the constituent agents are unmanned aerial vehicles (UAVs). Aerial platforms must negotiate a tightly coupled, three-dimensional operating space in which aerodynamic interactions, rapid attitude dynamics, and limited on-board energy severely constrain both sensing and actuation. Unlike ground robots, which can tolerate centimeter-scale errors without immediate catastrophic consequences, UAVs risk loss of separation, aerodynamic stall, or mid-air collision from deviations of only a few tens of centimeters. Under such conditions, the latency and bandwidth limitations of wireless communication can make naïvely flat coordination strategies untenable; yet a fully centralized controller would introduce single points of failure that compromise the very robustness sought in swarm solutions. A dynamically reconfigurable hierarchy offers a principled trade-off: by localizing high-rate control loops within sub-swarms while allowing global objectives to propagate through leader rotation and multi-level consensus, it can be used to preserve scalability and interchangeability without sacrificing the fast response demanded by aerial maneuvers (Vásárhelyi et al., 2018). This thesis proposes that dynamically reconfigurable

hierarchy is ideal for aerial swarms because they require not only the ability to create and reconfigure multi-level command structures autonomously, but also precise mechanisms for maintaining geometrically consistent formations and for preserving collective function when individual platforms fail or drop out of the network. These twin requirements—*formation control*, which guarantees spatial coherence under aerodynamic coupling and limited on-board sensing, and *fault tolerance*, which safeguards the mission against hardware faults, communication dropouts, or adversarial disturbances—constitute the core technical challenges tackled in contemporary UAV-swarm literature (Vásárhelyi et al., 2018; Zhou et al., 2022).

The following two sections examine the state of the art in formation control for aerial swarms and recent advances in fault-tolerant strategies that aim to endow large-scale UAV collectives with the robustness demanded by real-world deployments.

2.3 Formation control for aerial swarm robotics

Formation control is a fundamental coordination task in swarm robotics, where multiple robots move in unison to maintain a prescribed geometric pattern (Oh et al., 2015). This capability is important for aerial swarms for missions such as environmental monitoring, search-and-rescue, and surveillance of natural disaster sites (Bu et al., 2024). Over the years, researchers have developed various approaches to achieve and maintain formations in multi-robot systems (Oh et al., 2015). These methods differ in how the formation shape is specified and how the control decisions are made across the swarm. Common strategies can be grouped into three broad categories (Oh and Ahn, 2011):

- Behavior-based methods: Each robot follows simple local behaviors or rules (e.g., cohesion, separation, alignment) that collectively yield an emergent formation. This approach requires no explicit leader; instead, the formation arises from the interactions of individual behaviors.

- Leader–follower methods: One or a few designated robots act as leaders, and the remaining robots (followers) adjust their motions to maintain specified

positions (distances or angles) in relation to the leaders. The formation's shape is thus maintained by tracking leader trajectories.

- Virtual structure methods: The entire group is treated as a single rigid entity (a virtual structure). Control laws are designed as if commanding a large rigid body; each robot tracks a moving reference point on this imagined structure, preserving the overall formation geometry.

Each of these approaches has its own advantages and disadvantages in terms of flexibility, scalability, and the requirements for communication and sensing. Regardless of the specific strategy, a central challenge in formation control is maintaining the desired geometric shape of the formation as the swarm moves (De Queiroz et al., 2019). The robots must coordinate in such a way that their positions relative to each other remain as desired (including during translation or rotation of the whole formation). Achieving this robustly calls for tools to analyze and guarantee formation shape stability (De Queiroz et al., 2019). One key theoretical framework that addresses this need is graph rigidity theory, which provides conditions under which a formation's shape is preserved by inter-robot constraints.

In the context of multi-robot formations, graph rigidity theory models the formation as a graph: each robot is a vertex, and edges represent constraints between robots, typically desired distances. A formation (or framework) is said to be rigid if it cannot continuously deform into a different shape without breaking at least one of those inter-robot constraints (De Queiroz et al., 2019). Intuitively, a rigid formation behaves like a stiff structure: its inter-robot distances do not flex or collapse under external perturbations and can tolerate desired motions of the whole structure such as translating or rotating in space. The idea of rigidity stems from structural engineering (e.g. when studying a strut-and-tie truss structure) and has since been developed into a formal mathematical theory (De Queiroz et al., 2019). In recent years, it has played a fundamental role in formation control theory, especially for distance-constrained formations.

Formally, rigidity can be assessed through tools like rigidity matrix and rank conditions; for instance, a formation in plane requires a minimum number of fixed

constraints (e.g., distances) to be rigid, often characterized by Laman's theorem for generic configurations (a minimally rigid graph in 2D has $2n - 3$ edges for n robots) (Zhao and Zelazo, 2019). When a formation is infinitesimally rigid (a strong form of rigidity where even small instantaneous motions are constrained), distributed control laws can be proven to drive the robots to the desired shape and keep it there.

Classical rigidity theory deals primarily with distance constraints (often called distance rigidity). A distance-constrained formation is called rigid if the only way to move it without altering any of the specified inter-robot distances is by translating or rotating the entire group as if it were a single unit (Zhao and Zelazo, 2019). However, distances are not the only way to encode a formation shape. In some cases, especially relevant to vision-based sensing or when adaptive scaling of the formation is needed, using relative bearing (i.e. angle of arrival) information between robots is advantageous. This has led to an extension of the rigidity concept to formations defined by bearing measurements, known as bearing rigidity theory. Bearing rigidity theory is an analogous framework to distance rigidity, but instead of fixed distances, it considers fixed bearings between neighboring robots (Zhao and Zelazo, 2019). The central question it addresses is: under what conditions can the geometric pattern of a network of robots be uniquely determined if the bearing of each inter-robot link is specified? In a bearing-constrained formation, each edge of the interaction graph carries a desired vector direction rather than a desired length. If a formation is bearing rigid, then fixing the inter-neighbor angles uniquely determines the shape of the formation up to a scale factor. In other words, all robots can freely move closer or further apart as long as the lines of sight to their neighbors remain at the same angles, and such coordinated movement will be the only degree of freedom in the formation (Zhao and Zelazo, 2015).

Maintaining a formation via bearings is particularly useful in scenarios where distance measurements are unreliable or when the formation needs to adapt its overall size on the fly. Due to the local communication requirement of robot swarms (strictly based on neighbors directly exchanging local information; no global coordinator) (Brambilla et al., 2013), using relative bearings as the formation feedback is sensible – each robot can obtain the direction of its neighbor's signal or position

and does not need precise distance measurements or GPS. When considering robot swarms with self-organizing hierarchy, an additional benefit is that a bearing-based formation is robust to frequent adjustments in who follows whom (i.e., the graph connections rewiring as leadership roles change), because as long as the new interaction graph remains bearing rigid, the overall shape is preserved (Tang et al., 2022). In short, bearing rigidity provides a graceful way to enforce formation coherence amid self-organized hierarchical reconfiguration – the swarm's shape integrity will not be lost when the control network rearranges itself.

In summary, leveraging bearing rigidity can be a useful way for robot swarms to maintain stable formations in real-world scenarios: the shape of the formation is maintained by strict rigidity principles, and at the same time the swarm can take advantage of the flexibility of bearing-based control to adapt its size and configuration as needed. This capability is highly advantageous for aerial swarms operating in three-dimensional and often unpredictable flight environments—the swarm can, for example, contract and expand to pass through different narrow passageways and open areas, simultaneously re-organizing its internal hierarchy structure, and still maintain the relative geometry of the formation. Moreover, bearing-only sensing is a realistic mode for many robotic systems (e.g. using cameras or directional antennas, a robot can observe the direction of a neighbor but not its exact distance), and thus bearing rigidity aligns well with the limitations of onboard sensors on relatively simple robot platforms (Zhao and Zelazo, 2019).

2.4 Network routing protocols

In parallel with advances in multi-robot coordination and formation control, there is a long-standing body of work on multi-hop routing in mobile ad hoc networks (MANETs), wireless sensor networks (WSNs), and, more recently, flying ad hoc networks (FANETs). Classical MANET routing protocols are typically classified as *proactive* (table-driven), *reactive* (on-demand), or *hybrid*. Proactive protocols such as destination-sequenced distance vector (DSDV) and optimized link state routing (OLSR) maintain consistent and up-to-date routing tables at each node through

periodic dissemination of topology information, enabling low-latency forwarding at the cost of continuous control overhead and limited scalability under high mobility (Clausen and Jacquet, 2003; Perkins and Bhagwat, 1994). Reactive protocols such as ad-hoc on-demand distance vector (AODV) and dynamic source routing (DSR) instead discover routes only when needed by flooding route-request packets through the network and establishing reverse paths from the destination; this reduces control overhead in sparse or low-traffic regimes but incurs route discovery delays and can suffer from route instability in highly dynamic settings (Haas, 1997; Perkins et al., 2003). Hybrid approaches such as zone routing protocols (ZRP) aim to combine these advantages by maintaining a proactive state within a limited neighborhood while using on-demand mechanisms for longer-range communication (Samar et al., 2004).

Beyond single-path routing, a large literature has explored *multi-path* routing protocols that maintain several disjoint or partially disjoint routes between a given source and destination, with the goal of increasing reliability, throughput, or load balancing (Boukerche et al., 2011; Marina and Das, 2001). In these protocols, each node typically selects next hops based on composite link or path costs, which may include hop count, estimated delay, residual energy, congestion, or link-quality indicators. In FANET-oriented work, additional challenges such as rapid topology changes, directional antennas, and 3D motion have motivated the use of geographic/position-based routing, predictive link-quality estimation, and cross-layer designs (Bekmezci et al., 2013; Guillen-Perez and Cano, 2018). However, these schemes are largely developed at the network layer, where nodes are treated as generic relays. As a result, they make little use of the specific communication patterns and performance objectives that arise in multi-robot control tasks (e.g., formation keeping, consensus, or cooperative tracking), and may therefore optimize for generic metrics such as throughput or hop count rather than for closed-loop performance (e.g., bounded delay, synchronized updates, or robustness of the control law).

In robot swarms and multi-UAV systems research, the network layer is often abstracted away or modeled as an idealized broadcast channel. When routing is considered explicitly, standard MANET stacks are frequently adopted as off-the-

shelf components, or simple flooding/gossip mechanisms are used to disseminate information throughout the swarm (Guillen-Perez et al., 2021; Oubbati et al., 2019). These approaches provide basic connectivity but generally ignore the communication patterns implied by the swarm's control architecture—for example, leader–follower hierarchies or fixed neighbor relationships in a formation. Additionally, most routing protocols are designed to handle topological failures (e.g., broken links, node mobility) rather than informational faults such as corrupted sensor readings, intermittent estimation errors, or adversarial data injections that may propagate along otherwise healthy links. This focus is aligned with existing fault-tolerance research in swarm robotics, which typically studies robustness against individual agents permanently failing or dropping out, rather than disseminating incorrect information.

2.5 Fault tolerance in aerial swarm robotics

Reliability in networked systems requires consistently accurate information exchange among components, often under dynamic and uncertain conditions (Kirst et al., 2016; Zavlanos et al., 2011). If communication links fail or become unreliable during multi-hop communication, system convergence and performance guarantees can be compromised (Cortés, 2008; De Gennaro and Jadbabaie, 2006; Moreau, 2005; Olfati-Saber et al., 2007). In robot swarms, this challenge is enlarged by asynchronous ad-hoc communication and decentralized coordination of actuation and decision making. Robots in a swarm rely solely on local information and communication with nearby robots, without any estimation of the global state of the swarm or its environment, often leading to prolonged convergence times and vulnerability to the spread of incorrect information (Valentini et al., 2015). Frequent communication between robots can cause faulty information to spread quickly and potentially degrade overall swarm performance or lead to permanent failures.

Self-organized robot swarms exhibit some inherent fault tolerance, through redundancy and a lack of single points of failure (Dorigo et al., 2014; Hamann, 2018). However, many fault types are not mitigated by this passive tolerance and instead require dedicated mechanisms for detection and mitigation (Bjerknes and Winfield,

2013; Dorigo et al., 2021; Heinrich et al., 2022; Tarapore et al., 2017). Somewhat counter-intuitively, self-organized robot swarms are inherently much more tolerant to complete robot failures than to partial ones (Winfield and Nembrini, 2006). For example, a single robot producing faulty or malicious information has been shown to be capable of severe disruption to overall swarm behavior (Strobel et al., 2018; Winfield and Nembrini, 2006). Faulty robots can also physically obstruct the rest of the swarm, and this interference can paradoxically be worsened by the redundancy that provides swarms with some types of inherent fault tolerance (Pini et al., 2011).

Other faults to which self-organized robot swarms are vulnerable and which require dedicated mechanisms for detection and mitigation are *intermittent faults* (IFs). IFs are temporary faults that can appear, disappear, and reappear (Breitfelder and Messina, 2000), potentially caused by communication interference, sensor malfunctions, or software bugs (Zhou et al., 2019). IFs are difficult to detect and diagnose due to their transience (Niu et al., 2021) and can cause significant disruptions without leaving an easily detectable trace (Zhou et al., 2019). A representative example involves intermittent GPS signal degradation in cluttered environments, which can induce sporadic localization errors. These errors propagate through decentralized state estimation protocols, gradually undermining coordination mechanisms without generating explicit failure indicators. In real applications, e.g., in robot swarms deployed in inaccessible or dangerous environments (Dorigo et al., 2020, 2021), the consequences of IFs to mission performance and to safety can be severe and in some cases could be irreversible. Detecting and resolving IFs before they escalate is key to minimizing disruption: early detection can prevent cascading failures leading to erroneous execution of tasks and can prevent culmination in permanent failures, either of individual robots or the swarm as a whole (O'Keeffe and Millard, 2023).

IFs are difficult to detect in robot swarms with fully self-organized ad-hoc networks, because the network topology is transient and often unpredictable. IFs are much more straightforward to detect in fully centralized systems and in networks with static structures, for example in sensor networks (Sheng et al., 2021; Syed et al., 2016; Zhang et al., 2021). However, for multi-robot systems, full centralization and fully static networks also present downsides, such as single points of failure and limited

scalability.

Swarm robotics usually studies passive tolerance to *permanent faults*—that is, faults such as electromechanical failures that will remain unless they are actively repaired. When relying on passive fault tolerance, studies have usually demonstrated that a swarm continues its mission after some or many robots have failed, either by continuing with fewer robots (Bjerknes and Winfield, 2013; Khadidos et al., 2015; Rubenstein et al., 2014) or by replacing/repairing the failed robots without pausing the mission (Christensen et al., 2009; Mathews et al., 2017; Varadharajan et al., 2020; Zhu et al., 2024).

Swarm robotics studies that focus specifically on fault tolerance do not typically rely on passive tolerance, instead developing dedicated mechanisms to handle permanent faults. The majority of these methods detect and react to permanent electromechanical failures after they have occurred (Millard, 2016; O'Keeffe and Millard, 2023), often relying on time-out mechanisms in which a robot is considered non-operational if it does not respond to a message within a certain time. Existing methods for detecting permanent faults include LED synchronization (Christensen et al., 2009), simulation comparison (Millard et al., 2014), shared sensor data analysis (Khadidos et al., 2015), and behavioral feature vectors (BFVs) (Tarapore et al., 2017). These methods often focus on detection, assuming that once a fault is detected, a repair or other intervention is possible during normal operation (e.g., (Christensen et al., 2009; O'Keeffe and Millard, 2023; Oladiran, 2019)). Although such repairs might be unrealistic in inaccessible, hazardous, or congested environments (Dorigo et al., 2021; O'Keeffe and Millard, 2023; Tarapore et al., 2019), future methods for autonomous repair could be developed to complement detection. In short, the existing reactive methods can be considered effective for many types of permanent faults (Bjerknes and Winfield, 2013). However, these detection approaches are unlikely to be applicable to the transience of IFs and their long response times (Khaldi et al., 2017) would likely be too slow for the early detection and recovery that IFs require. Methods to detect and repair IFs in robot swarms still need to be developed.

There are no existing swarm robotics methods focused on IF detection and recovery. Strategies developed for IFs in other types of systems, such as model-based anal-

ysis (e.g., discrete-event-system models (Carvalho et al., 2017), causal models (Abdelwahed et al., 2008)) and quantitative analysis (e.g., parameter estimation (Zhang et al., 2020), geometric approaches (Yan et al., 2018), Kalman-like filtering (Zhang et al., 2018)), provide valuable insights but primarily target single-unit systems with static and known system models (Syed et al., 2016; Yaramasu et al., 2015), which is incompatible with self-organized systems such as robot swarms. Likewise, IF strategies developed for sensor networks (Sheng et al., 2021; Zhang et al., 2021) typically use fully centralized architectures to correct information transmission and reception (Syed et al., 2016), and are therefore incompatible with self-organized systems.

Furthermore, although fully centralized monitoring is highly effective for detecting and correcting IFs, it can present problems of inflexibility, limited scalability, and single points of failure (e.g., at the point where monitoring is centralized). Fully self-organized approaches, by contrast, would be highly flexible and offer greater scalability and a lack of single points of failure, but would present problems of limited accuracy and potentially slow reaction times.

Fault tolerance is particularly crucial in aerial swarms. Unlike ground-based robots that can withstand momentary lapses in control or communication, aerial robots must maintain constant flight stability to avoid catastrophic failures (Fourlas and Karras, 2021; Nguyen and Pitakwachara, 2024). Faults affecting state estimation or actuator commands can rapidly escalate, as any loss of lift or orientation in one robot might lead to collisions or uncontrolled descents, posing safety risks to both the swarm and its surroundings. Furthermore, communication within aerial swarms typically relies on wireless links that are prone to interference, signal attenuation, and bandwidth constraints, making consensus mechanisms even more susceptible to dropped messages or delayed updates (Baktayan et al., 2024; Huang and Huang, 2025). Energy limitations compound these vulnerabilities—each robot operates under strict battery and payload constraints, so additional resources spent on compensating for a malfunctioning neighbor or rerouting communication can quickly deplete reserves, compromising mission duration and reliability (Alam and Moh, 2022; Beigi et al., 2022). Environmental unpredictability adds another layer of complexity, as flights in unstructured or cluttered spaces can expose robots to wind disturbances

or sporadic sensor noise. Under such conditions, failures in localization or sensing can easily propagate erroneous state estimates throughout the swarm, compromising its coordination and causing widespread performance loss or, in the worst case, group-level instability or full system failure (Power et al., 2020).

Chapter 3

S-drone: Aerial Robotic Platform

This chapter[1] *presents an open-source UAV platform–S-drone for swarm robotics research. Most research involving UAVs in swarm robotics presents only simulation results, while key landmark studies with real UAV swarms have used UAV platforms that were custom-built for the respective study. One important reason for this is that no commercial UAV platform comes pre-equipped with the ability to identify and track the positions and poses of nearby drones using only onboard sensors and computation, and in research platforms, the relevant sensing technologies are currently under development. Our aim is to provide a platform that allows swarm robotics researchers to test their algorithms on real UAVs, without having to develop their own custom-built UAVs or to wait until more advanced sensing technology is ready off-the-shelf. We provide a well-documented, entirely open-source UAV platform—S-drone (Swarm-drone)—to foster and support UAV swarm research in a laboratory environment. The S-drone uses fiducial markers in the environment and cooperative feature-level sensor fusion for indirect inter-robot tracking. That is, each UAV estimates the presence, identity, relative 2D position, and relative 2D orientation of neighboring peers indirectly from the observed AprilTags. The S-drone is suitable for a wide range of contexts, supports quad-camera vision-based navigation and a variety of onboard sensing, and is extensible. It is especially suited for swarm robotics research because it can operate using strictly onboard processing and sensing without the need for external positioning systems or ground stations for off-board sensing.*

[1]The content of this chapter was previously published in (Oğuz et al., 2022) and (Oğuz et al., 2024). Sections 3.7 and 3.8 were included in the supplementary materials of (Zhu et al., 2024).

3.1 Approach and contributions

The *S-drone* for swarm robotics research (see Fig. 3.1) is designed with the objectives to: support both piloted and autonomous flight, support the development of control algorithms for deployments ranging from a simple single UAV to complex missions with UAV swarms, and operate without an external positioning system or support from a ground station. Based on these objectives, we prioritized the following features when designing the *S-drone* platform:

- **Robot-to-robot coordination and vision-based navigation.** The *S-drone* has onboard localization, navigation, and coordination capabilities for multi-robot operations. Also, its quad-camera configuration gives it a wide enough field of view at low altitudes to support vision-based navigation indoors.

- **Decentralization.** It can operate without any external centralized infrastructure such as global positioning systems, motion capture systems, off-board sensing, off-board processing, or ground control stations.

- **Simulator.** The *S-drone* comes with an open-source plug-in for simulation in ARGoS (Pinciroli et al., 2012), a multi-physics engine simulator for robot swarms.

- **Transferability from simulation to real hardware.** The *S-drone* plug-in for ARGoS provides a high-level control interface, enabling users to implement the same control software in both simulation and on real hardware.

- **Extensible control.** By using ARGoS and the PX4 flight controller, it is possible to develop and implement a wide range of new swarm algorithms and control laws.

- **Open-source.** All hardware, software, and mechanical design files, as well as assembly instructions, modeling and flight control design, and operation

instructions, are hosted on Open Science Framework[2].

- **Accessible and extensible hardware.** Most of the components are commercially available, and those that are not can be manufactured (either in-house or by a third-party service provider) using the provided design files and widely available machinery (e.g., 3D printer). The *S-drone* is also customizable. It uses well-modularized components when possible and it is straightforward to integrate a variety of off-the-shelf sensors (e.g., a localization module for GPS-guided outdoor navigation or a 360° LiDAR module for mapping). Deeper customization can also be made using the design files provided (e.g., mechanical mounts can be customized to change the sensor types or fields of view).

- **Software support for managing experiments.** The *S-drone* is supported by an experiment *Supervisor* program, used for starting, monitoring, and shutting down real UAV experiments.

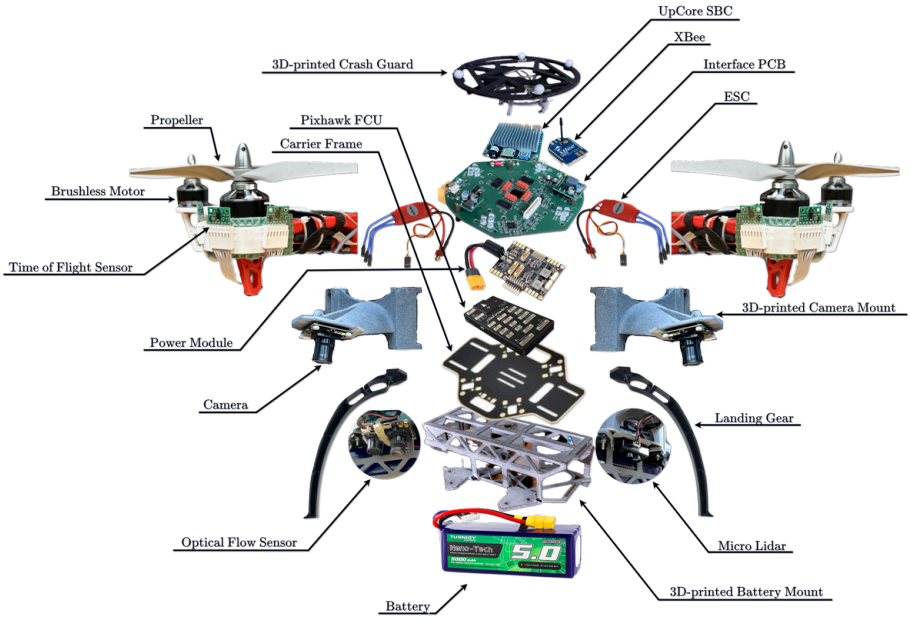
Based on these design priorities, we equipped the *S-drone* with a combination of standard UAV components, other off-the-shelf components, and a few custom components.

First, the standard UAV components include many that are essential for the autonomous or piloted flight of UAV platforms, including electronic speed control (ESC) modules, brushless motors, propellers, a single-point LiDAR, an optical flow module, a battery, and a flight controller. These standard components are off-the-shelf from the hobbyist market and therefore are widely available, affordable, well-modularized, and well-interfaced.

Second, the other off-the-shelf components are those that have not been produced specifically for UAV platforms. The *S-drone* uses these components for autonomous flight and decentralized coordination. They are: the cameras, the single-board computer, and the optional attachment for collision avoidance using time-of-flight (ToF) distance sensors. The *S-drone* is also equipped with two separate WiFi communication links: one for low-level interaction with the UAV (e.g., controlling and monitor-

[2] https://osf.io/hp6rf/

(a)



(b)

Figure 3.1: (a) A set of four *S-drones*. (b) Exploded view of the *S-drone*. (Reprinted from (Oğuz et al., 2024).)

ing the Pixhawk4 flight controller) and one for communication with the single-board computer (e.g., interaction with Linux or ARGoS).

Third, some custom components are necessary. Many versions of the off-the-shelf components exist, but not all combinations are compatible, and because many of them are not modularized for UAV platforms, they need to be interfaced. Also, the components overall are interconnected and constrain each other in a complex way. For these reasons, we designed a custom printed circuit board to interface the components. The *S-drone* interface printed circuit board (PCB) includes power modules and connectors for peripherals, powers the flight controller and other components, and provides reliable data communication between the onboard computer and the flight controller and other components. The *S-drone* also includes custom cables and some custom mechanical components.

Built with these components, the *S-drone* is a quadrotor based on the DJI F450 airframe, with a diagonal motor-to-motor wheelbase of 450 mm (corresponding to approximately 0.32 m rotor spacing along the roll and pitch axes), an overall footprint of about 0.32×0.32 m in the horizontal plane, an overall height of about 0.25 m when resting on its landing gear (excluding propellers), a total take-off weight of approximately 2.5 kg (including battery and sensors). Its basic operational characteristics are:

- a maximum flight time of approximately 15 minutes in hover (as built, without extensions),
- a maximum velocity of 25 m/s,
- an operational angular velocity of 220°/s on roll and pitch axes and 15°/s on the yaw axis,
- a maximum altitude while flying autonomously without external infrastructure (i.e., by relying on its downward-facing single-point LiDAR and optical flow sensor) of approximately 12 m,
- can detect tags at a rate of five frames per second (FPS) with an input resolution of 700×700 pixels when flying at an altitude of 1 m and using the default

AprilTag and camera configuration.

We also aim to supply an approach for local inter-UAV tracking that is straightforward for swarm robotics researchers to use on our *S-drone* or implement on another UAV platform. We have therefore included in our *S-drone* platform an open-source approach for local inter-UAV tracking in laboratory environments that does not require specialist sensor technology nor expertise. The onboard capability requirements for our approach are: downward-facing visual camera(s), a single-board computer, and the ability to send messages between specific UAVs when in close physical proximity.

The *S-drone* is equipped for autonomous vision-based waypoint navigation, using AprilTags (Olson, 2011) in the environment (see Sec. IV for demonstrations). The AprilTag detection used for individual waypoint navigation is also used for inter-UAV tracking. When two *S-drone*s have the same AprilTag in their field of view, they can detect each other indirectly, using feature-level cooperative sensor fusion between UAVs (see Fig. 3.2). Effectively, each UAV knows its own ID and can detect the IDs and relative poses of any AprilTags in its onboard cameras' fields of view. Then, each UAV broadcasts the IDs of the AprilTags it currently detects, along with its own ID. If a UAV receives a message that contains an AprilTag ID matching one of those it currently detects, it unicasts the feature-level information it has about the respective AprilTag to the ID of the respective message sender. Thus, when two UAVs detect the same AprilTag, they both receive the pose information of that AprilTag, relative to the other UAV's body frame. Using this information, each UAV can calculate the pose of the other UAV, relative to its own body frame. The *S-drone* platform is equipped with this simple cooperative sensor fusion and includes all software components needed for operation. We demonstrate inter-UAV tracking with the *S-drone* in scenarios of multi-UAV formations and cooperative object transport.

In the remainder of this section, we describe the details of the *S-drone*'s electronics, mechanical components, and software.

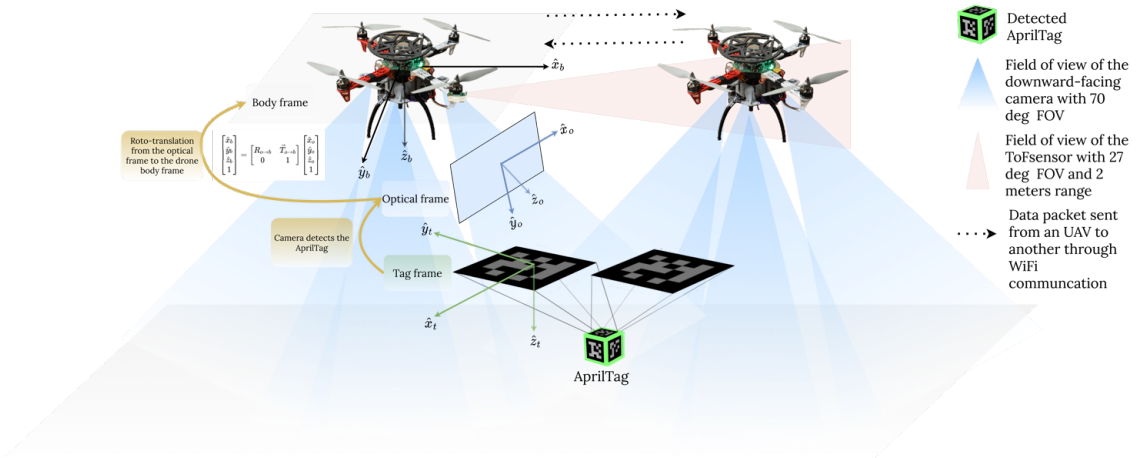


Figure 3.2: Illustration of tag-based localization and cooperative sensor fusion in the *S-drone*. When a UAV detects an AprilTag, it rotates and translates the tag's relative pose information from the camera's optical frame to the UAV body frame. If two UAVs simultaneously detect the same AprilTag, they exchange information about the tag. Using this exchanged information, each UAV calculates the relative pose of the other UAV, with respect to its own body frame. The field-of-view coverage of the onboard ToF range sensors is also depicted for completeness; however, these ToF sensors are not used in the subsequent experiments presented in this work. For details, please see the open-source repository. (Based on a figure from (Oğuz et al., 2024)

.)

3.2 Electronic components

The electronic components of the *S-drone* can be categorized in four groups: 1) the flight control electronics, 2) the interface PCB, 3) the propulsion electronics, and 4) the autonomy electronics. The connectivity diagram of the components can be seen in Fig. 3.3.

3.2.1 Flight control electronics

The *S-drone*'s flight control electronics are responsible for controlling the UAV's attitude, position, and trajectory. These components are all commercially available for modularized UAV platforms.

In the *S-drone*, we use a Pixhawk4 flight controller from the manufacturer Holybro. The Pixhawk4 is based on an open-source project (Meier et al., 2011) and is optimized to run the open-source autopilot software called PX4. Its primary purpose is to fly and control the UAV without constant 'hands-on' remote control by a human operator being required. It has a central STM32F765 processor, an I/O STM32F100 processor, an accelerometer, a gyroscope, a magnetometer, barometric air pressure sensors, five serial ports, three I2C ports, and four SPI buses. In the *S-drone*, the Pixhawk4 is connected to the interface PCB via a serial port. Through the interface PCB, it communicates with the Up Core at 921 600 baud rate and 80 000 bytes/second data rate. The Pixhawk4 is operated at 4.9-5.5 V voltage and is powered from an external PM07 power module, which is manufactured specifically for the Pixhawk and also manages power distribution to the ESC modules for the brushless motors. The PM07 power module also sends information to the Pixhawk4 about the battery voltage and current supplied to the motors.

For relative position estimation, the *S-drone* includes a downward-facing PX4Flow optical flow smart camera module from Holybro. To help in precision hovering, terrain following, and precision landing, the *S-drone* also includes a downward-facing TFMini single-point LiDAR from Benawake. The optical flow sensor captures 752×480 pixels images and estimates the UAV's velocity at 400 Hz. The single-point LiDAR is low-cost and low-power, has a range of 12 m, and provides measurement

data at 100 Hz at a baud rate of 115 200. These sensors work indoors and outdoors without the need for an external supplement, e.g., external illumination for the flow sensor.

3.2.2 Interface PCB

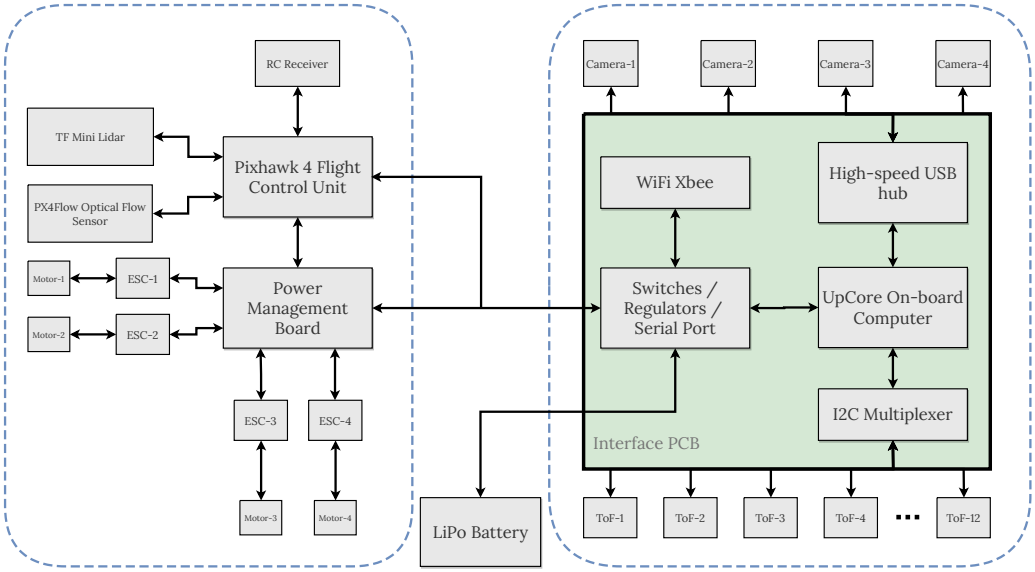
The *S-drone* has a two-sided interface PCB that allows communication between electronic components and provides connectors for peripherals. Here we describe the main components of the PCB (see Fig. 3.3).

The *S-drone* also includes some removable components to optionally support manual piloting, which can be important, e.g., during development phases, even when working with swarm algorithms for autonomous drones. Using a DSMX remote receiver from Spektrum, an operator can remotely control the UAV's movement (e.g., speed, direction, throttle, yaw angle, etc.), switch between the flight modes, or engage a kill-switch in an emergency. Using SiK telemetry radios from Holybro, which provide a wireless data link between a ground control station and the Pixhawk4, an operator can adjust control gains and review telemetry data in real-time, which is useful for development phases.

3.2.3 Propulsion electronics

The propulsion system of a UAV is crucial: it determines key performance metrics such as flight time, payload capacity, flying speed, and range. The propulsion electronics of the *S-drone* comprise motors, ESC modules, and a battery. Note that, although there are many possible versions of these components, only a few combinations are compatible. Poor combinations would result in UAVs that either fail under system stress or cannot take off at all.

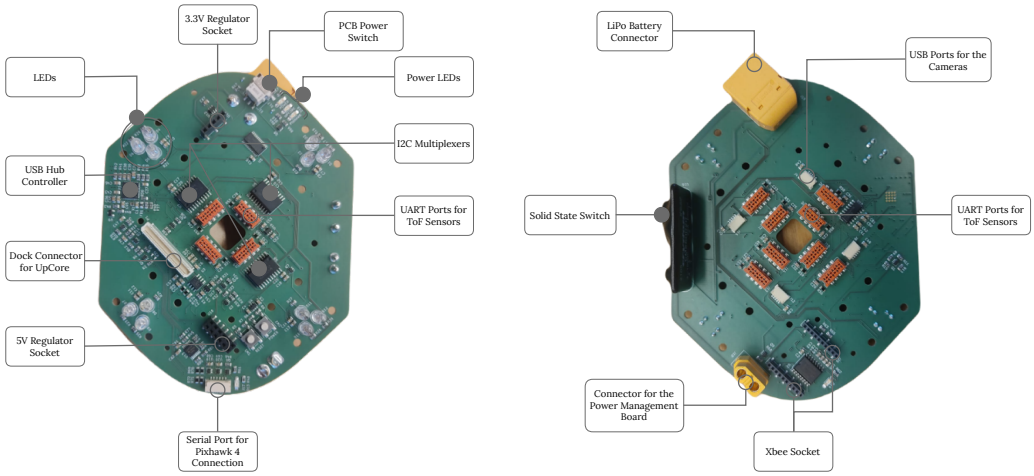
The *S-drone* has four MN-series brushless DC motors from T-Motor. The current fed to the motors is controlled by four 30 A ESC modules with SimonK firmware. The ESC modules control the speed of the motors based on the PWM signal from the Pixhawk4. Finally, the *S-drone* is powered by a lithium-ion polymer (LiPo) 5000 mAh four-cell battery.



Flight and actuator control electronics

Interface PCB and autonomy electronics

(a)



The top face of the interface PCB

The bottom face of the interface PCB

(b)

Figure 3.3: (a) Connectivity diagram for the electronics of the *S-drone*. (b) The interface PCB. (Reprinted from (Oğuz et al., 2024).)

On the top face of the PCB, located next to the LiPo battery connector, there is a power switch to turn the PCB power on and off. The top face also has two step-down voltage regulators (a 3.3 V regulator for the control circuitry and a 5 V regulator for the Up Core) and a docking connector for the Up Core. On its bottom face, the PCB includes a standard docking socket for an XBee WiFi module, which communicates with a router in a 2.4 GHz frequency band. For experiment management (e.g., triggering the start of an experiment or engaging a kill-switch for safety), this module provides a two-way communication link between the experiment *Supervisor* software and the UAV. Both sides of the PCB also include connectors for autonomy electronics (refer to Sec. II.A.4): USB ports with custom JST connectors for cameras and UART ports for ToF distance sensors with 27-degree field-of-view flash LiDAR. To orchestrate the data coming from many ToF sensors, the PCB includes three low-voltage four-channel I2C multiplexers, which are each responsible for sending the distance information from up to four ToF sensors to the Up Core via I2C (i.e., supporting up to twelve ToF sensors in total).

On the top face of the PCB there are also 12 through-hole round RGB LEDs and four surface mounted diode (SMD) LEDs. The through-hole RGB LEDs are driven by a 16-bit LED driver from NXP, used to set the brightness of the RGB channels and for color mixing. The colors of the through-hole RGB LEDs can provide visual state indicators for a developed swarm algorithm and the colors of the surface-mounted LEDs provide visual feedback about the on/off status of the PCB, Pixhawk4, Up Core, XBee WiFi module, and *S-drone* autonomous mode.

3.2.4 Autonomy electronics

The autonomy electronics of the *S-drone* comprise an Up Core single-board computer, four downward-facing cameras, and an optional attachment for collision avoidance using twelve ToF distance sensors with 27-degree field-of-view flash LiDAR. While the field-of-view coverage of these ToF sensors is documented for completeness, this collision-avoidance attachment is not used in the experiments reported in this thesis. Similar ToF-based range-and-bearing sensing has been employed in micro air vehicle

swarms such as in (Bilaloğlu et al., 2022). Note that the attachment for collision avoidance is considered optional (i.e., not required for flight) because the *S-drone* can operate without it, and the attachment is designed to be easily mechanically interchangeable with other custom sensor attachments. Substitutions can be made by customizing the provided open-access design files.

The Up Core single-board computer is responsible for processing sensing and control information, as needed for autonomy and swarm algorithms. It communicates with the flight controller, the cameras, and extensions for further autonomy such as the ToF distance sensors. The Up Core has an Intel Atom X5 Z8350 processor, an Intel Gen. 8 HD Graphics 400 GPU, 4 GB DDR3L RAM, 64 GB eMMC, two USB 2.0 ports, one UART port, and an embedded WiFi module.

The cameras of the *S-drone* are used for autonomous vision-based navigation and detection of other robots during decentralized coordination. The camera modules are manufactured by Leopard Imaging, are based on a 5 MP OV5650 image sensor from OmniVision, and have a 2.8 mm manual focus lens. The pixel data from the image sensors are routed from these connectors to the low-power USB 2.0 hub controllers called USB4715 from Microchip. Then, the USB hub controller routes the pixel data to the Up Core's USB 2.0 data port.

The optional ToF distance sensors are intended to increase the situational awareness capabilities of the *S-drone* for purposes such as collision avoidance. The attachment includes twelve VL53LOX flash LiDAR ToF distance sensors from Pololu. They have a composite field of view that covers most of the circumference directions of the UAV and can measure the distance of an object at up to 2 meters.

3.3 Mechanical components

The *S-drone* includes several 3D printed components, which have been designed according to mechanical stiffness, anti-vibration, and manufacturing constraints. The components have been designed to reduce volume as much as possible (and therefore reduce both weight and manufacturing costs) while maintaining appropriate mechanical properties.

It is important to note that the camera mounts need to be carefully designed to meet anti-vibration requirements. The UAV is relatively heavy and therefore the motors are quite powerful and produce strong vibrations. Also, the cameras need to be mounted sufficiently away from the body of the UAV in order to have an unobstructed view, effectively creating cantilever beams, which can easily deflect with significant magnitude on the unsupported end (i.e., the end with the camera). Any customization of the *S-drone* will need to take this consideration into account.

3.4 Software components

The software components for the *S-drone* allow for transferring code from simulation to real hardware and support simulations using the accompanying simulator ARGoS. The seamless transition from simulation to real hardware enables swarm algorithms to be developed in the simulation environment before deploying them on real UAVs, reducing the amount of testing on real hardware and the potential for crashes during the development phase. The full firmware and support software are provided in the Open Science Framework repository.

The *S-drone* uses the Yocto Project[3] to automatically build a custom embedded Linux distribution at runtime. In the Yocto build system, layers define a specific version and configuration of the Linux kernel alongside userspace configuration and software. We use a Docker image to partially automate the configuration of the build system and to provide a clean and consistent environment for the build process.

The custom layer `meta-drone` for the Yocto build system includes a Linux kernel based on Linux Yocto 5.2 with several patches.[4] The layer `meta-drone` also contains configuration segments that are combined to form the `defconfig` for the kernel, which results in the necessary kernel modules being compiled and installed, and the ACPI configuration to communicate the location and configuration of all the devices

[3] https://www.yoctoproject.org

[4] Patches are an upstream version of PCA954x I2C multiplexer driver for compatibility with ACPI (Advanced Configuration and Power Interface), a workaround for USB video class (UVC) bandwidth calculation (so that multiple cameras can operate on the same bus), and an update to the VL53l0X ToF driver to incorporate sampling based on the industrial input-output (IIO) kernel API.

to the Linux kernel drivers.

The *S-drone* preinstalled software includes LibIIO (a userspace library for the kernel industrial input-output API), Intel's threading building blocks (TBB) library, the AprilTag library, MJPEG streamer, the PX4 flight system library, and Python 3 (with packages for libjpeg-turbo, pymavlink, pySerial, and V4L2). We have also included a custom JSON-based RPC service called Fernbedienung,[5] which allows us to run programs remotely and interact with them over the standard input, output, and error streams. This service is written in Python and is started automatically on boot using `Systemd`.

For running experiments, we use an executable control interface of the UAV defined in the ARGoS simulator. By using this executable and the ARGoS libraries, we are able to use the same control software in simulation as we do on real UAVs. The executable can be passed configuration settings (e.g., identifiers, message router configuration settings) as arguments and provides an implementation of all of the sensors and actuators defined in the control interface for the *S-drone*. To give a few examples, the flight system is implemented using the PX4 library; the camera system is implemented using the V4L2, Intel TBB, and AprilTag libraries; the ToF distance sensors are implemented using LibIIO; and the LEDs are implemented on top of the sysfs virtual filesystem.

We also use a custom *Supervisor* program for starting, monitoring, and shutting down multi-robot experiments. The program provides a web-based user interface in which the user can record data from an Optitrack tracking system, log messages sent between robots, and capture ARGoS's standard output and standard error from each UAV during an experiment. The *Supervisor* program has been written in Rust. The back-end is built on top of the Tokio asynchronous framework/runtime and the front-end (compiled to WebAssembly for the browser) is built on top of the Yew web framework. The front-end and back-end communicate with each other over a WebSocket.

For *S-drone* controllers, and for the development of swarm algorithms or other types of control, the *S-drone* uses Lua control software, including Lua scripts which

[5] https://github.com/iridia-ulb/fernbedienung-python

run on top of the Lua bindings for the sensors and actuators defined in the control interface. By writing our controllers in Lua, there is no need for recompilation each time a change is made to our control software. Also, because Lua has a small footprint and is fast and portable, it can be easily embedded, avoiding potential problems with, e.g., incompatible instruction sets or linking libraries. Experiments with the *S-drone* use an XML configuration file to define the sensors and actuators in use and the update rate of the controller.

3.5 Mathematical modeling

The preliminary information needed for the UAV modeling includes the reference frames, the rotation matrix for transformations between reference frames, the UAV states, and the sensor and motor information.

3.5.1 Reference frames

The *S-drone* is modeled using the body frame (denoted B) and the inertial frame (denoted $\mathcal{I}$), as shown in Fig. 3.4.

The body frame is a relative coordinate system that represents the body of the *S-drone* . The origin of the frame is the *S-drone* center of mass, the x-axis of the frame is the *S-drone* roll axis (i.e., longitudinal axis, directed to the front), the y-axis is the pitch axis (i.e., transverse axis, directed to the right), and the z-axis is the yaw axis (i.e., vertical axis, directed to the bottom).

The inertial frame is a fixed coordinate system defined at an arbitrary point on the Earth's surface and can be defined at any point on the surface. The inertial frame uses a North-East-Down (NED) configuration, in which the x-axis is directed northward, the y-axis is directed eastward, and the z-axis is directed downward.

To transform vectors defined in the body frame B into the inertial frame $\mathcal{I}$, we construct a zyx rotation matrix using Euler angles (roll, pitch, yaw). A zyx rotation involves three rotations (see Fig. 3.5): the frame is rotated around the z-axis (yaw rotation), then around the y-axis (pitch rotation), and lastly around the x-axis (roll rotation). The zyx rotation matrix ${}^{\mathcal{I}}\boldsymbol{R}_{\mathcal{B}}(\Phi, \Theta, \Psi) = \boldsymbol{R}_z(\Psi)\boldsymbol{R}_y(\Theta)\boldsymbol{R}_x(\Phi)$ is defined

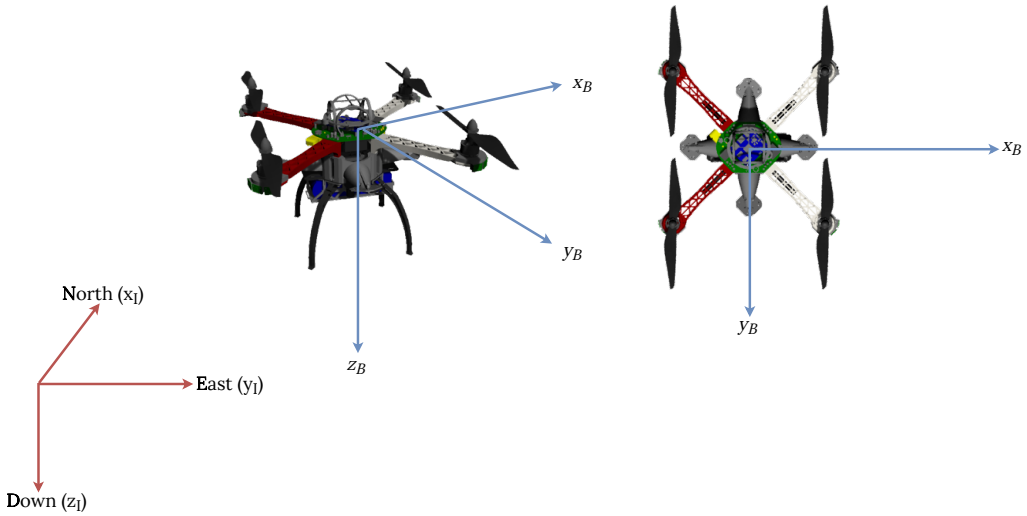


Figure 3.4: Reference frames. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

as:

$$
{}^{\mathcal{I}}\boldsymbol{R}_{\mathcal{B}}(\Phi,\Theta,\Psi) = \begin{bmatrix} \cos\Psi\cos\Theta & \cos\Psi\sin\Theta\sin\Phi - \sin\Psi\cos\Phi & \cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi \\ \sin\Psi\cos\Theta & \sin\Psi\sin\Theta\sin\Phi + \cos\Psi\cos\Phi & \cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi \\ -\sin\Theta & \sin\Phi\cos\Theta & \cos\Phi\cos\Theta \end{bmatrix} \tag{3.1}
$$

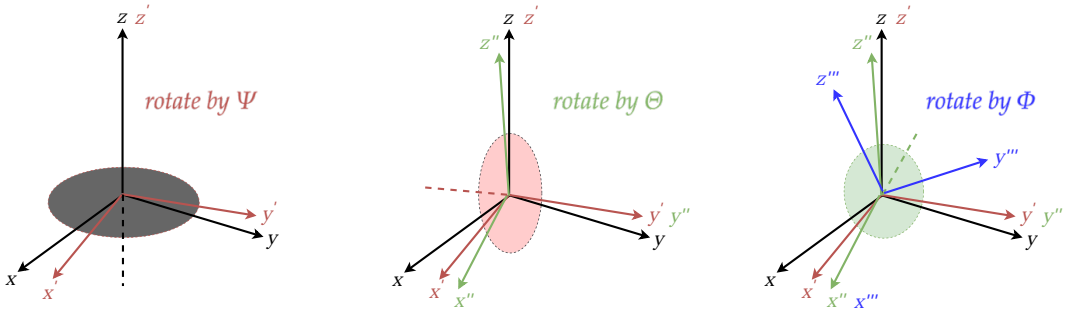


Figure 3.5: The zyx rotation configuration. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

3.5.2 State variables

The *S-drone* has both inertial and body frame states, which can be used to describe its position, attitude, and velocity (see Table 3.1). The inertial frame states, which are defined relative to a fixed reference point, include the UAV's position coordinates (x, y, z), its rotational angles (ϕ, θ, ψ), and its linear and angular velocities $(\dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi})$. The body frame states, which are defined relative to the UAV itself, include the body frame linear velocities (u, v, w) and body frame angular velocities (p, q, r). These states are important for understanding the UAV's motion and internal dynamics.

Vector	Description
$\boldsymbol{\zeta}_I = \begin{bmatrix} x & y & z \end{bmatrix}$	Positions in the inertial frame
$\boldsymbol{\eta}_I = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}$	Euler angles (roll, pitch, yaw) in the inertial frame
$\boldsymbol{V}_I = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}$	Linear velocities in the inertial frame
$\boldsymbol{\omega}_I = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}$	Angular velocities in the inertial frame
$\boldsymbol{V}_B = \begin{bmatrix} u & v & w \end{bmatrix}$	Linear velocities in the body frame
$\boldsymbol{\omega}_B = \begin{bmatrix} p & q & r \end{bmatrix}$	Angular velocities in the body frame

Table 3.1: State vectors.

3.5.3 Sensor models

To control the *S-drone*, it is necessary to observe the states. The simulation model of the UAV uses the following measurement models for the inertial sensors: a three-axis accelerometer, a three-axis gyroscope, and a three-axis magnetometer.

Remark 1: The *S-drone* does not rely on external localization information, such as GPS signals, so we do not consider those types of sensors. Instead, we use inertial sensors in our UAV to measure the platform's orientation, velocity, and acceleration.

These sensors provide independent and non-jammable measurements of the platform, as described in (Farrell, 2008).

The gyroscope measures angular rotation around the body frame axis by measuring the Coriolis Force, which acts on objects in a rotating reference frame (Farrell, 2008). In our measurement model, we assume that the gyroscope measures body frame angular velocities (p, q, r) directly, which can be integrated over time to compute the orientation of the sensor (ϕ, θ, ψ). The measurement model for the gyroscope is as follows:

$$\begin{aligned} \boldsymbol{\omega}_B^m &= \boldsymbol{\omega}_B + \boldsymbol{b}(t) + \boldsymbol{\mu} \\ \dot{\boldsymbol{b}}(t) &= \boldsymbol{\nu}_g \mathbf{1}_{3\times 1} \end{aligned} \tag{3.2}$$

where $\boldsymbol{\omega}_B^m \in \mathbb{R}^3$ represents the measurement value of the angular velocities (p, q, r) in the body frame, the slowly time-varying bias term $\boldsymbol{b}(t) \in \mathbb{R}^3$ changes with white Gaussian noise $\boldsymbol{\nu}_g$, $\boldsymbol{\mu} \in \mathbb{R}^3$ is the measurement noise term, and $\mathbf{1}_{3\times 1}$ is a matrix of 1s in the shape 3×1 matrix.

The accelerometer detects the forces present and uses them to calculate acceleration, according to the mass of the object and D'Alembert's force principle (Farrell, 2008). Our measurement model for the accelerometer determines the forces acting on the UAV and calculates its acceleration as follows:

$$\begin{aligned} \boldsymbol{a}_B^m &= \boldsymbol{a}_B^l + {}^{\mathcal{B}}\boldsymbol{R}_{\mathcal{I}} \boldsymbol{a}_I^g \boldsymbol{b}(t) + \boldsymbol{\mu} \\ \dot{\boldsymbol{b}}(t) &= \boldsymbol{\nu}_a \mathbf{1}_{3\times 1} \end{aligned} \tag{3.3}$$

where $\boldsymbol{a}_B^l \in \mathbb{R}^3$ is the linear acceleration vector in the body frame, $\boldsymbol{a}_I^g \in \mathbb{R}^3$ is the gravitational acceleration vector in the inertial frame, the slowly time-varying bias term $\boldsymbol{b}(t) \in \mathbb{R}^3$ changes with the white Gaussian noise $\boldsymbol{\nu}_a$, and $\boldsymbol{\mu} \in \mathbb{R}^3$ is the measurement noise term.

The magnetometer measures the strength and direction of the earth's magnetic field, is used to help the UAV maintain a stable hover, and is used in the autonomous navigation and positioning processes happening onboard. Our measurement model

for the magnetometer is:

$$\begin{aligned} \boldsymbol{H}_B^m &= {}^{\mathcal{B}}\boldsymbol{R}_{\mathcal{I}}\boldsymbol{H}_I + \boldsymbol{\mu} \\ \boldsymbol{H}_I = \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix} &= \|\boldsymbol{H}_I\| \cdot \begin{bmatrix} \cos\beta \\ 0 \\ \sin\beta \end{bmatrix} \end{aligned} \tag{3.4}$$

where $\boldsymbol{H}_I \in \mathbb{R}^3$ is the magnetic field vector and β is the magnetic inclination. If the measurements of the roll and pitch angles are already known, the yaw angle can then be calculated as:

$$\psi = \arctan \frac{H_x^m \cos\theta + (H_y^m \sin\phi + H_z^m \cos\phi \sin\theta)}{H_x^m \sin\phi - H_y^m \cos\phi} \tag{3.5}$$

3.5.4 Actuator dynamics

We represent the dynamics of the motors using a first-order transfer function (Bouabdallah and Siegwart, 2007), which allows for modeling the time-varying behavior of the motors and thereby accurately predicting the response of the UAV to control inputs. The transfer function describes the relationship between the input and output of the motor and can be used to design control algorithms that accurately regulate the speed and torque of the motors. The model is given as

$$\frac{\Omega}{\Omega_d} = \frac{1}{T_{rot}s + 1} \tag{3.6}$$

where T_{rot} is a time constant, Ω_d is the desired motor speed, and Ω is the calculated motor speed.

3.5.5 Nonlinear system representation

For our nonlinear model of the *S-drone*'s dynamic behavior, we make the following assumptions, as in (Bouabdallah and Siegwart, 2007), to simplify some of the modeling calculations:

- the UAV body structure is rigid,
- the UAV is symmetrical in all axes,
- the propeller structure is rigid and the oscillation on the UAV body does not show motion in the vertical direction,
- the body frame coincides with the UAV's center of gravity,
- motors are identical,
- motors are positioned perpendicular to the body frame,
- propeller thrust and drag moment are directly proportional to the square of the motor speed,
- the ground effect is neglected,
- the gyroscopic effect of the motors is neglected.

3.5.6 Kinematic models

To study the motion of the quadrotor using kinematics, we use the rotation matrix (3.1) to transform linear velocities measured in the body frame to the inertial frame, as follows:

$$\begin{aligned} \dot{\boldsymbol{\zeta}}_I = \boldsymbol{V}_I = {}^{\mathcal{I}}\boldsymbol{R}_{\mathcal{B}}\boldsymbol{V}_B \\ {}^{\mathcal{B}}\boldsymbol{R}_{\mathcal{I}} = ({}^{\mathcal{I}}\boldsymbol{R}_{\mathcal{B}})^T \end{aligned} \tag{3.7}$$

Then, to transform the angular velocities from the body frame to the inertial frame, we use the angular transformation matrix ${}^{\mathcal{I}}\boldsymbol{T}_{\mathcal{B}}$, defined as

$${}^{\mathcal{I}}\boldsymbol{T}_{\mathcal{B}} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \quad \text{and} \quad {}^{\mathcal{B}}\boldsymbol{T}_{\mathcal{I}} = ({}^{\mathcal{I}}\boldsymbol{T}_{\mathcal{B}})^T \tag{3.8}$$

We can then transform the angular velocities from the body frame to the inertial frame, as

$$\dot{\boldsymbol{\eta}}_I = {}^{\mathcal{I}}\boldsymbol{T}_{\mathcal{B}}\,\boldsymbol{\omega}_B \tag{3.9}$$

and write the quadrotor kinematics as

$$\begin{bmatrix}\dot{\boldsymbol{\zeta}}_I \\ \dot{\boldsymbol{\eta}}_I\end{bmatrix} = \begin{bmatrix}{}^{\mathcal{I}}\boldsymbol{R}_{\mathcal{B}} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & {}^{\mathcal{I}}\boldsymbol{T}_{\mathcal{B}}\end{bmatrix}\begin{bmatrix}\boldsymbol{V}_B \\ \boldsymbol{\omega}_B\end{bmatrix} \tag{3.10}$$

where $\mathbf{0}_{3\times3}$ is a matrix of 3s in the shape 3×3 and

$$\dot{\boldsymbol{\zeta}}_I = \begin{cases}\dot{x} &= w[\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta] - v[\cos\phi\sin\psi + \cos\psi\sin\phi\sin\theta] + u[\cos\psi\cos\theta] \\ \dot{y} &= -w[\cos\psi\sin\phi - \cos\phi\sin\psi\sin\theta] + v[\cos\phi\cos\psi + \sin\phi\sin\psi\sin\theta] + u[\cos\theta\sin\psi] \\ \dot{z} &= w[\cos\phi\cos\theta] + v[\cos\theta\sin\phi] - u\sin\theta\end{cases} \tag{3.11}$$

and

$$\dot{\boldsymbol{\eta}}_I = \begin{cases}\dot{\phi} &= p + r[\cos\phi\tan\theta] + q[\sin\phi\tan\theta] \\ \dot{\theta} &= q\cos\phi - r\sin\phi \\ \dot{\psi} &= r\frac{\cos\phi}{\cos\theta} + q\frac{\sin\phi}{\cos\theta}\end{cases}. \tag{3.12}$$

3.5.7 Dynamic models

For the dynamics of the quadrotor, we need to consider the mass and inertia, the forces, and the torques acting on the body.

To derive the differential equations describing the quadrotor dynamics using the Newton-Euler method (Fossen, 1994), we start with the following equalities:

$$\begin{bmatrix}\boldsymbol{F}_B \\ \boldsymbol{\tau}_B\end{bmatrix} = \begin{bmatrix}m\boldsymbol{I}_3 & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \boldsymbol{J}\end{bmatrix}\begin{bmatrix}\dot{\boldsymbol{V}}_B \\ \dot{\boldsymbol{\omega}}_B\end{bmatrix} + \begin{bmatrix}\boldsymbol{\omega}_B \times m\boldsymbol{V}_B \\ \boldsymbol{\omega}_B \times \boldsymbol{J}\boldsymbol{V}_B\end{bmatrix} \tag{3.13}$$

$$\boldsymbol{F}_B = \begin{cases} F_x &= m(\dot{u} + qw - rv) \\ F_y &= m(\dot{v} - pw - ru) \\ F_z &= m(\dot{w} + pv - qu) \end{cases}, \quad \boldsymbol{\tau}_B = \begin{cases} \tau_x &= \dot{p}J_x - qrJ_y + qrJ_z \\ \tau_y &= \dot{q}J_y + prJ_x - prJ_z \\ \tau_z &= \dot{r}J_z + pqJ_x + pqJ_y \end{cases}$$

where $\boldsymbol{I}_3$ is a 3×3 identity matrix, $\boldsymbol{J}$ is a 3×3 diagonal inertia matrix, m is the mass of the quadrotor, and $\boldsymbol{F}_B = \begin{bmatrix} F_x & F_y & F_z \end{bmatrix}^T \in \mathbb{R}^3, \boldsymbol{\tau}_B = \begin{bmatrix} \tau_x & \tau_y & \tau_z \end{bmatrix}^T \in \mathbb{R}^3$ are force and torque vectors that act on the body. The force vector ($\boldsymbol{F}_B$) is composed of the gravitational force and the thrust force generated by the propulsion system. Similarly, the torque vector ($\boldsymbol{\tau}_B$) is composed of the roll, pitch, and yaw moments as well as the gyroscopic moments generated by the propulsion system. They can be expressed as

$$\begin{aligned} \boldsymbol{F}_B &= \overbrace{mg\,{}^{\mathcal{B}}\boldsymbol{R}_{\mathcal{I}}.\hat{\boldsymbol{e}}_z}^{\text{the gravity effect}} - \overbrace{U_1.\hat{\boldsymbol{e}}_3}^{\text{the thrust force}} \\ \boldsymbol{\tau}_B &= \underbrace{\boldsymbol{U}_\tau}_{\text{the moment vector}} - \underbrace{\boldsymbol{\tau}_g}_{\text{the gyroscopic moment}} \end{aligned} \tag{3.14}$$

where g is the gravitational acceleration, $\hat{\boldsymbol{e}}_z$ is the z-axis unit vector in the inertial frame, $\hat{\boldsymbol{e}}_3$ is the z-axis unit vector in the body frame, and the total thrust force U_1 (i.e., the combination of forces F_1, F_2, F_3, and F_4) provides the lift necessary for the quadrotor to ascend or descend (see Fig. 3.6). The moment vector $\boldsymbol{U}_\tau = \begin{bmatrix} U_2 & U_3 & U_4 \end{bmatrix}^T$ includes moments U_2, U_3, and U_4. U_2 and U_3 are generated through the arms of the quadrotor and correspond to roll and pitch moments, respectively, whereas U_4 is the total yaw moment, which is the combination of moments M_1, M_2, M_3, and M_4 (see Fig. 3.6). The gravitational force mg acting on the quadrotor is defined in the inertial frame and transformed to the body frame using the rotation matrix ${}^{\mathcal{B}}\boldsymbol{R}_{\mathcal{I}}$. In addition to these forces, the quadrotor's propulsion system generates a gyroscopic moment $\boldsymbol{\tau}_g$, which arises due to the change in the direction of the angular momentum vector of the motors as the quadrotor rolls and pitches, that can be expressed as

$$\boldsymbol{\tau}_g = \sum_{i=1}^{4} \boldsymbol{J}(\boldsymbol{\omega}_B \times \hat{\boldsymbol{e}}_3)(-1)^{i+1}\Omega_i \tag{3.15}$$

where $\boldsymbol{J}$ is the inertia of the motors and is one of the moments acting on the quadro-

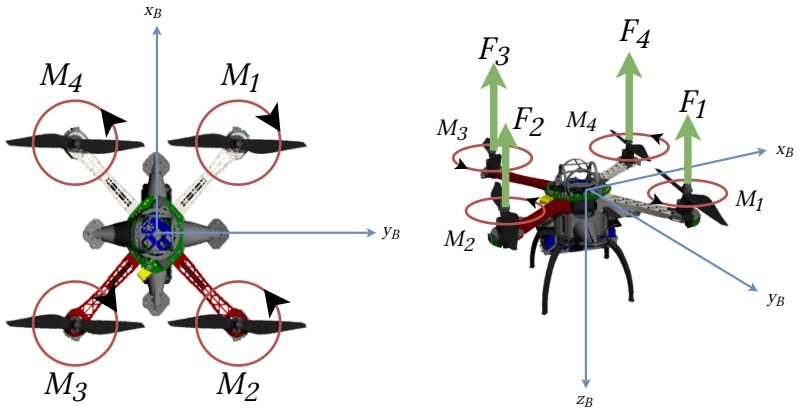


Figure 3.6: The moments and forces acting on the quadrotor body. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

tor, and Ω_i is the angular velocity of the i-th motor. Because two of the motors on the quadrotor rotate clockwise and the other two rotate counterclockwise, the total angular velocity of the motors can be expressed as $\Omega_T = \Omega_1 - \Omega_2 + \Omega_3 - \Omega_4$. Then, we can rewrite the gyroscopic moment (3.15) as

$$\boldsymbol{\tau}_g = \boldsymbol{J} \left(\begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ \Omega_T \end{bmatrix} \right) \tag{3.16}$$

The motor inertia $\boldsymbol{J}$ is minimal compared to the other moments acting on the quadrotor (Bouabdallah and Siegwart, 2007). Therefore, when building the model of the quadrotor, it is generally safe to neglect the gyroscopic effect of the motors, the ground effect during takeoff or landing, the dynamic effects of the propellers' airflow and blade flapping, or other aerodynamic effects that might be observed during flight. Using these simplifications, and under the assumptions previously mentioned, we can

construct the dynamics model of the quadrotor as follows:

$$\begin{aligned} -mg\sin\theta &= m(\dot{u} + qw - rv) \\ mg\cos\theta\sin\phi &= m(\dot{v} - pw - ru) \\ mg\cos\theta\cos\phi - U_1 &= m(\dot{w} + pv - qu) \\ U_2 &= \dot{p}J_x - qrJ_y + qrJ_z \\ U_3 &= \dot{q}J_y + prJ_x - prJ_z \\ U_4 &= \dot{r}J_z + pqJ_x + pqJ_Y \end{aligned} \tag{3.17}$$

In this model, U_1 is the total thrust force and only acts in the z-axis direction of the body frame. Assuming K_T is a constant thrust coefficient, U_1 can be defined as

$$U_1 = \begin{bmatrix} 0 \\ 0 \\ K_T(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \end{bmatrix} \tag{3.18}$$

such that the total thrust force is a function of the square of the rotational speed of the motors. U_2 is the roll angular moment, which is generated about the x-axis of the body frame, and is defined as

$$U_2 = l_x K_T(-\Omega_1^2 - \Omega_2^2 + \Omega_3^2 + \Omega_4^2)\sin 45^\circ \tag{3.19}$$

where l_x is the moment arm distance of the motor frame from the x-axis of the body frame.

U_3 is the pitch angular moment, which is generated about the y-axis of the body frame, and is defined as

$$U_3 = l_y K_T(\Omega_1^2 - \Omega_2^2 - \Omega_3^2 + \Omega_4^2)\sin 45^\circ \tag{3.20}$$

where l_y is the moment arm distance of the motor frame from the y-axis of the body frame.

U_4 is the yaw angular momentum, which is generated about the z axis of the

body frame, and is defined as

$$U_4 = K_M(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \tag{3.21}$$

where K_M is a constant moment coefficient that reflects the inherent moment-generating capabilities of the motor-propeller system. Note that there is no moment arm coefficient included here because, in our quadrotor, there is no distance between the point at which the moment is applied and the yaw axis.

By combining equations (3.18), (3.19), (3.20), and (3.21), the forces and moments generated by the propulsion system of the quadrotor can be obtained as

$$\begin{bmatrix} U_1 \\ \boldsymbol{U}_\tau \end{bmatrix} = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} K_T & K_T & K_T & K_T \\ -K_T l_x \frac{\sqrt{2}}{2} & -K_T l_x \frac{\sqrt{2}}{2} & -K_T l_x \frac{\sqrt{2}}{2} & -K_T l_x \frac{\sqrt{2}}{2} \\ K_T l_y \frac{\sqrt{2}}{2} & -K_T l_y \frac{\sqrt{2}}{2} & -K_T l_y \frac{\sqrt{2}}{2} & K_T l_y \frac{\sqrt{2}}{2} \\ K_M & -K_M & K_M & -K_M \end{bmatrix} \begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} \tag{3.22}$$

The inverse of the transformation matrix in Eq. (3.22) can then be used to determine the motor angular velocities needed to produce the desired forces and moments, as follows:

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{4K_T} & -\frac{1}{4K_T l_x} & \frac{1}{4K_T l_y} & \frac{1}{4K_M} \\ \frac{1}{4K_T} & -\frac{1}{4K_T l_x} & -\frac{1}{4K_T l_y} & -\frac{1}{4K_M} \\ \frac{1}{4K_T} & \frac{1}{4K_T l_x} & -\frac{1}{4K_T l_y} & \frac{1}{4K_M} \\ \frac{1}{4K_T} & \frac{1}{4K_T l_x} & \frac{1}{4K_T l_y} & -\frac{1}{4K_M} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{3.23}$$

3.5.8 State-space representation

To construct the quadrotor model we use for control and simulation, we now express the model in state space form, which combines the kinematic and dynamic equations. To create the state space representation of the quadrotor's model, we define the following state vector:

$$\boldsymbol{X} = [\ x \ \ y \ \ z \ \ u \ \ v \ \ w \ \ \phi \ \ \theta \ \ \psi \ \ p \ \ q \ \ r\ ]^T \tag{3.24}$$

Then, the complete model of the quadrotor can be written using the equation sets in Eq. (3.11) (3.12) and (3.17), as

$$\dot{\boldsymbol{X}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} w(\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta) - v(\cos\phi\sin\psi + \cos\psi\sin\phi\sin\theta) + u\cos\psi\cos\theta \\ -w(\cos\psi\sin\phi - \cos\phi\sin\psi\sin\theta) + v(\cos\phi\cos\psi + \sin\phi\sin\psi\sin\theta) + u\cos\theta\sin\psi \\ w\cos\phi\cos\theta + v\cos\theta\sin\phi - u\sin\theta \\ rv - qw - g\sin\theta \\ pw - ru + g\cos\theta\sin\phi \\ qu - pv + g\cos\theta\cos\phi - \frac{U_1}{m} \\ p + r\cos\phi\tan\theta + q\sin\phi\tan\theta \\ q\cos\phi - r\sin\phi \\ r\frac{\cos\phi}{\cos\theta} + q\frac{\sin\phi}{\cos\theta} \\ qr\frac{I_y - I_z}{I_x} + \frac{U_2}{I_x} \\ pr\frac{I_z - I_x}{I_y} + \frac{U_3}{I_y} \\ pq\frac{I_x - I_y}{I_z} + \frac{U_4}{I_z} \end{bmatrix} \tag{3.25}$$

The model given in Eq. (3.25) includes states that are defined in the body frame, namely translational acceleration states ($\dot{u},\ \dot{v},\ \dot{w}$) and rotational acceleration states ($\dot{p},\ \dot{q},\ \dot{r}$). Because it is more convenient to work with calculations in the inertial frame for control studies (Bouabdallah and Siegwart, 2007), we redefine the body frame states of the state vector in the inertial frame. To express the translational states ($\dot{u}, \dot{v}, \dot{w}$) in the inertial frame, we apply the rotation matrix ${}^{\mathcal{I}}\boldsymbol{R}_{\mathcal{B}}$ as follows:

$$\begin{aligned} \boldsymbol{F}_B &= m\begin{bmatrix} \dot{u} & \dot{v} & \dot{w} \end{bmatrix}^T \\ \boldsymbol{F}_B &= mg{}^{\mathcal{B}}\boldsymbol{R}_{\mathcal{I}}.\hat{\boldsymbol{e}}_z - U_1.\hat{\boldsymbol{e}}_3 \\ {}^{\mathcal{I}}\boldsymbol{R}_{\mathcal{B}}\boldsymbol{F}_B &= mg\hat{\boldsymbol{e}}_z - U_1{}^{\mathcal{I}}\boldsymbol{R}_{\mathcal{B}}\hat{\boldsymbol{e}}_3 \end{aligned} \tag{3.26}$$

Then we can obtain the translational states in the inertial frame, denoted as $(\ddot{x}, \ddot{y}, \ddot{z})$:

$$\begin{aligned}
{}^{\mathcal{I}}\boldsymbol{R}_{\mathcal{B}}\boldsymbol{F}_B &= mg\hat{\boldsymbol{e}}_z - U_1{}^{\mathcal{I}}\boldsymbol{R}_{\mathcal{B}}\hat{\boldsymbol{e}}_3 \\
m\dot{\boldsymbol{V}}_I &= mg\hat{\boldsymbol{e}}_z - U_1{}^{\mathcal{I}}\boldsymbol{R}_{\mathcal{B}}\hat{\boldsymbol{e}}_3 \\
m\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} &= mg\hat{\boldsymbol{e}}_z - U_1{}^{\mathcal{I}}\boldsymbol{R}_{\mathcal{B}}\hat{\boldsymbol{e}}_3 \\
\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} &= g\hat{\boldsymbol{e}}_z - \frac{U_1{}^{\mathcal{I}}\boldsymbol{R}_{\mathcal{B}}\hat{\boldsymbol{e}}_3}{m} \\
&= \begin{bmatrix} -\frac{U_1}{m}(\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta) \\ -\frac{U_1}{m}(-\cos\psi\sin\phi + \cos\phi\sin\psi\sin\theta) \\ g - \frac{U_1}{m}\cos\phi\cos\theta \end{bmatrix}
\end{aligned} \tag{3.27}$$

To express the rotational states $(\dot{p},\ \dot{q},\ \dot{r})$ in the inertial frame, we establish the relationship between the body frame and the inertial frame, using equation (3.9), as

$$\begin{bmatrix}p\\ q\\ r\end{bmatrix} = {}^{\mathcal{I}}\boldsymbol{T}_{\mathcal{B}}\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}, \quad {}^{\mathcal{I}}\boldsymbol{T}_{\mathcal{B}} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}. \tag{3.28}$$
$$\begin{bmatrix}p\\ q\\ r\end{bmatrix} = \begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}$$

Remark 2: In the equation, (3.28), we make the assumption that the quadrotor primarily hovers or moves with small angles, meaning that the roll and pitch angles are approximately zero. This assumption, known as the small-angle approximation (Bouabdallah and Siegwart, 2007), allows us to simplify the angular transformation matrix given ${}^{\mathcal{I}}\boldsymbol{T}_{\mathcal{B}}$ to a 3×3 identity matrix.

With the relationship established and the corresponding equalities for $(\dot{p}, \dot{q}, \dot{r})$ in

Eq. (3.25), we can obtain the transformed rotational states $(\ddot{\phi}, \ddot{\theta}, \ddot{\psi})$ as

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \dot{\psi}\dot{\theta}\frac{I_y - I_z}{I_x} + \frac{U_2}{I_x} \\ \dot{\phi}\dot{\psi}\frac{I_z - I_x}{I_y} + \frac{U_3}{I_y} \\ \dot{\phi}\dot{\theta}\frac{I_x - I_y}{I_z} + \frac{U_4}{I_z} \end{bmatrix} \tag{3.29}$$

Then, the model in Eq. (3.25) becomes

$$\begin{aligned} \dot{\boldsymbol{X}} &= g(\boldsymbol{X}, \boldsymbol{U}) \\ &= f(\boldsymbol{X}) + \sum_{i=1}^{4} \boldsymbol{a}_i(\boldsymbol{X}) U_i \end{aligned} \tag{3.30}$$

where

$$\begin{aligned} \boldsymbol{X} &= [\ x \ \ y \ \ z \ \ \phi \ \ \theta \ \ \psi \ \ \dot{x} \ \ \dot{y} \ \ \dot{z} \ \ \dot{\phi} \ \ \dot{\theta} \ \ \dot{\psi}\ ]^T \\ \dot{\boldsymbol{X}} &= [\ \dot{x} \ \ \dot{y} \ \ \dot{z} \ \ \dot{\phi} \ \ \dot{\theta} \ \ \dot{\psi} \ \ \ddot{x} \ \ \ddot{y} \ \ \ddot{z} \ \ \ddot{\phi} \ \ \ddot{\theta} \ \ \ddot{\psi}\ ]^T \\ \boldsymbol{f}(\boldsymbol{X}) &= [\ \dot{x} \ \ \dot{y} \ \ \dot{z} \ \ \dot{\phi} \ \ \dot{\theta} \ \ \dot{\psi} \ \ 0 \ \ 0 \ \ g \ \ (\dot{\psi}\dot{\theta})\frac{I_y - I_z}{I_x} \ \ (\dot{\phi}\dot{\psi})\frac{I_z - I_x}{I_y} \ \ (\dot{\phi}\dot{\theta})\frac{I_x - I_y}{I_z}\ ]^T \end{aligned} \tag{3.31}$$

and

$$\begin{aligned} \boldsymbol{a}_1 &= [\ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ a_1^7 \ \ a_1^8 \ \ a_1^9 \ \ 0 \ \ 0 \ \ 0\ ]^T \\ a_1^7 &= -\frac{1}{m}(\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta) \\ a_1^8 &= -\frac{1}{m}(-\cos\psi\sin\phi + \cos\phi\sin\psi\sin\theta) \\ a_1^9 &= -\frac{1}{m}\cos\phi\cos\theta \\ \boldsymbol{a}_2 &= [\ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ \frac{1}{I_x} \ \ 0 \ \ 0\ ]^T \\ \boldsymbol{a}_3 &= [\ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ \frac{1}{I_y} \ \ 0\ ]^T \\ \boldsymbol{a}_4 &= [\ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ \frac{1}{I_z}\ ]^T \end{aligned} \tag{3.32}$$

3.5.9 Linearized system representation

In order to design a linear flight controller, for instance, a proportional–integral–derivative (PID) controller, it is necessary to linearize the model (3.30) around an operating point that brings the system to an equilibrium state. This process is described, following (Koo and Sastry, 1998), as

$$\dot{\boldsymbol{X}} = g(\bar{\boldsymbol{X}}, \bar{\boldsymbol{U}}) \rightarrow \mathbf{0} \tag{3.33}$$

where the control input vector $\boldsymbol{U} = \begin{bmatrix} U_1 & U_2 & U_3 & U_4 \end{bmatrix}^T$ consists of the total thrust U_1 and control moments. $\bar{\boldsymbol{X}}$ represents an operating point with a constant input $\bar{\boldsymbol{U}}$, which is known as the trim condition and is defined as

$$\bar{\boldsymbol{U}} = \begin{bmatrix} mg & 0 & 0 & 0 \end{bmatrix}^T \tag{3.34}$$

where mg represents the total thrust required to counteract the inertial force along the $+z$-axis (due to gravity and to the weight of the quadrotor) and maintain the hover state of the quadrotor.

To get the linearized model of Eq. (3.30) around the operating point $\bar{\boldsymbol{X}} = [\ \bar{x} \quad \bar{y} \quad \bar{z} \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\ ]^T$ with the constant input $\bar{\boldsymbol{U}}$, we use Taylor expansion as follows:

$$\dot{\boldsymbol{X}} = g(\bar{\boldsymbol{X}}, \bar{\boldsymbol{U}}) + \frac{\delta g}{\delta \boldsymbol{X}}|_{\bar{\boldsymbol{X}}} \delta \boldsymbol{X} + \frac{\delta g}{\delta \boldsymbol{U}}|_{\bar{\boldsymbol{U}}} \delta \boldsymbol{U} + \frac{1}{2} \frac{\delta^2 g}{\delta^2 \boldsymbol{X}}|_{\bar{\boldsymbol{X}}} \delta^2 \boldsymbol{X} + \frac{1}{2} \frac{\delta^2 g}{\delta^2 \boldsymbol{U}}|_{\bar{\boldsymbol{U}}} \delta^2 \boldsymbol{U} + \dots \tag{3.35}$$

where $\delta \boldsymbol{X}$ and $\delta \boldsymbol{U}$ represent deviations from the trim condition, with $\delta \boldsymbol{X} = \boldsymbol{X} - \bar{\boldsymbol{X}}$ and $\delta \boldsymbol{U} = \boldsymbol{U} - \bar{\boldsymbol{U}}$.

To simplify the Taylor expansion, we can ignore higher-order terms and consider only the first-order terms. Since $g(\bar{\boldsymbol{X}}, \bar{\boldsymbol{U}}) \rightarrow 0$, Eq. (3.35) becomes:

$$\dot{\boldsymbol{X}} \approx \frac{\partial g}{\partial \boldsymbol{X}}\bigg|_{\bar{\boldsymbol{X}}} \delta \boldsymbol{X} + \frac{\partial g}{\partial \boldsymbol{U}}\bigg|_{\bar{\boldsymbol{U}}} \delta \boldsymbol{U} \approx \boldsymbol{A}\boldsymbol{X} + \boldsymbol{B}\boldsymbol{U} \tag{3.36}$$

where

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \end{bmatrix}$$

3.6 Control architecture

The overall structure of the flight control system for the simulation studies is shown in Fig. 3.7. This system is based on the controllers used in the real quadrotor. We use a cascaded control structure that effectively decouples the position (position and linear velocity) and attitude (angle and angular velocity) control loops.

3.6.1 Position control

We use a position controller with a cascaded P/PID structure, with the first stage being a 3D position controller and the second stage being a linear velocity controller. Note that the quadrotor position control system is underactuated (Hua et al., 2013), therefore, the position control for the xy-axes can be achieved through attitude control. The position controller uses a P controller and its outputs for the xy axes produce the desired acceleration values $\ddot{x}^d$, $\ddot{y}^d$ in the inertial frame. The outputs of the position controller for the xy axes produce the desired acceleration values $\ddot{x}^d$, $\ddot{y}^d$ in the inertial frame. These values are then used to generate the desired roll ϕ^d

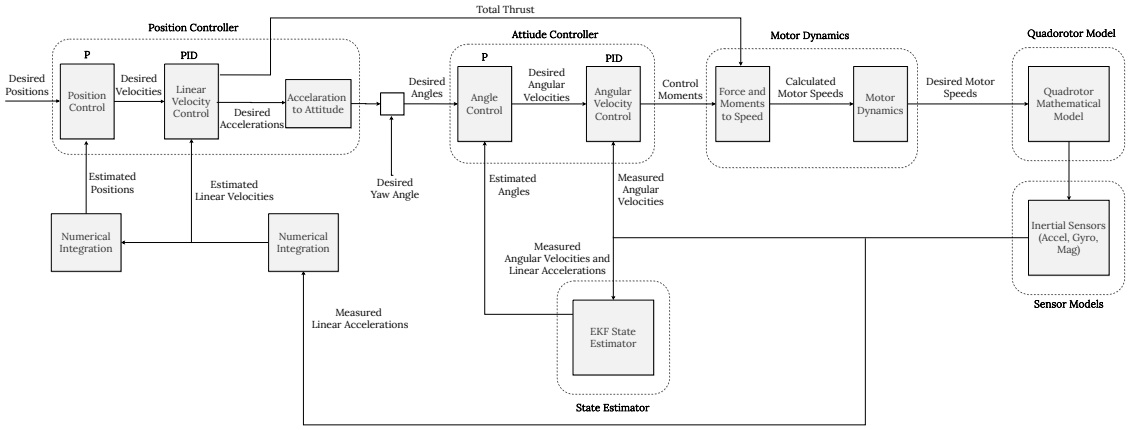


Figure 3.7: The flight control system. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

and pitch θ^d angles for the attitude controller to use. The overall structure of the position controller (see Fig. 3.8) is given as

$$\begin{aligned}
\textbf{Altitude control}: U_1 &= K_{Pz} e_{Vz} + K_{Iz} \int e_{Vz} - K_{Dz} \hat{\dot{z}} + mg \\
\textbf{Desired roll and pitch angles}: \phi^d &= \frac{m}{U_1}(-\ddot{x}^d \sin\hat{\psi} + \ddot{y}^d \cos\hat{\psi}) \\
\theta^d &= \frac{m}{U_1}(-\ddot{x}^d \cos\hat{\psi} - \ddot{y}^d \sin\hat{\psi}) \\
\text{where} \quad \ddot{x}^d &= K_{Px} e_{Vx} + K_{Ix} \int e_{Vx} - K_{Dx} \hat{\dot{x}} \\
\ddot{y}^d &= K_{Py} e_{Vy} + K_{Iy} \int e_{Vy} - K_{Dy} \hat{\dot{y}}
\end{aligned} \tag{3.37}$$

3.6.2 Attitude control

We use an attitude controller with a cascaded P/PID (proportional/proportional-integral-derivative) structure, with the first stage being an angle controller and the second stage being an angular velocity controller.

The angle controller uses a P controller that produces an output signal proportional to the error between the desired and actual values. The output of the angle

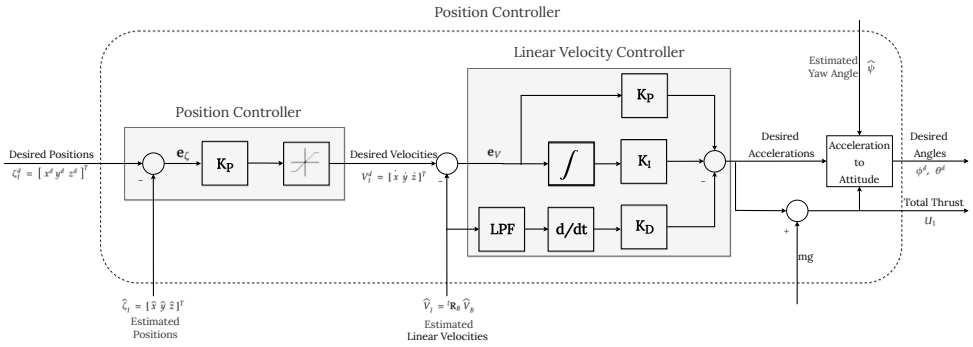


Figure 3.8: The cascaded position controller. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

controller becomes the desired value for the angular velocity controller, which uses a PID controller.

The outputs of the angular velocity controller, U_2, U_3, and U_4, are the control moments that are used to manipulate the quadrotor's attitude. The overall structure of the attitude controller (see Fig. 3.9) is given as

$$\begin{aligned}
&\textbf{Roll control}: U_2 = K_{P\phi} e_{\omega\phi} + K_{I\phi} \int e_{\omega\phi} - K_{D\phi} \dot{\hat{\phi}} \\
&\textbf{Pitch control}: U_3 = K_{P\theta} e_{\omega\theta} + K_{I\theta} \int e_{\omega\theta} - K_{D\theta} \dot{\hat{\theta}} \\
&\textbf{Yaw control}: U_4 = K_{P\psi} e_{\omega\psi} + K_{I\psi} \int e_{\omega\psi} - K_{D\psi} \dot{\hat{\psi}}
\end{aligned} \tag{3.38}$$

These controllers are used for both the real and simulated quadrotors.

3.7 Simulation and real experiment environment

We conduct our simulated experiments in the ARGoS multi-robot simulator (Pinciroli et al., 2012), a widely used simulator for swarm robotics research (see Fig. 3.10).

In ARGoS, sensors and actuators are plug-ins that have either read-only access or the ability to modify specific entities in the simulated 3D space (Pinciroli et al., 2012). In our experimentation setups, to control robots in a way that is replicable in simulation, we use the libraries and robot models of ARGoS as an interface to the

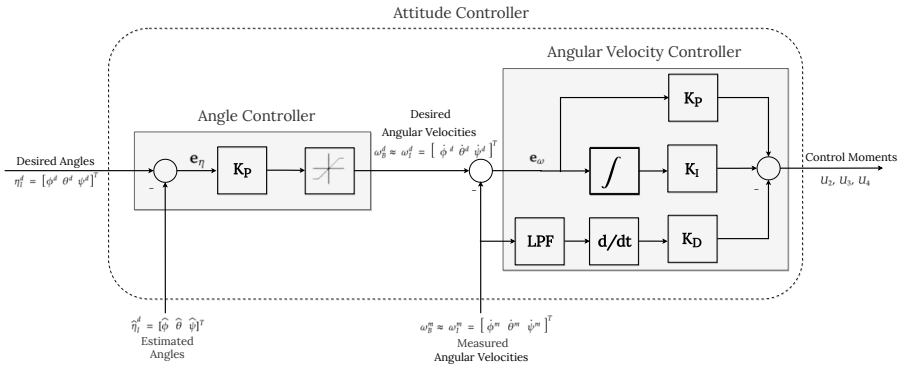


Figure 3.9: The cascaded attitude controller. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

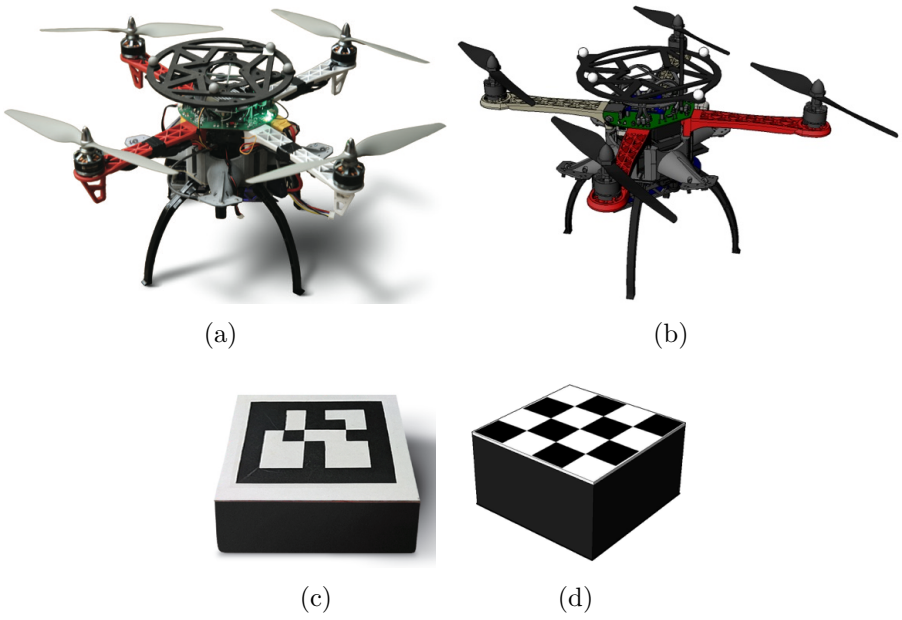


Figure 3.10: The (a) real and (b) simulated aerial robot. A (e) real and (f) simulated physical obstacle; not to size. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

low-level control of the robots. Specifically, we use executables based on the libraries of ARGoS that we have custom-developed for this study. These executables initialize the sensors and actuators of the robots in such a way that the provided high-level control interface matches that of the sensor and actuator plug-ins of our robot models in the ARGoS simulator. In other words, each robot's control interface provided by the ARGoS libraries is an abstraction layer on top of the physical hardware which is also used to create simulation models of the robots in ARGoS. Therefore, we can run exactly the same control software on both the real robots and the simulated robots, using C++ and the Lua scripting language.

For the models of the robot sensors and actuators in simulation, we conducted a trial-and-error calibration process to tune the speed and noise parameters. The speed parameters of the robot actuators in simulation have been tuned so that the speeds of the real robots match those of the simulated robots when receiving the same target linear and angular velocities output by the control algorithms. This calibration is straightforward, thus the resulting maximum and average speeds of the simulated and real robots are equivalent when receiving the same target velocities. By contrast, noise is influenced by many (often unknown) factors and therefore the noise parameters are much more challenging to calibrate and the result often underestimates the noise present in reality. In our setup, we tuned noise parameters of the quadrotor gyroscope and accelerometer.

To assess the noise displayed in multi-robot missions with interactive behaviors (i.e., those used in our experiments), we measure the difference in error in the results of our real robot experiments and the results of matching simulated setups. The difference is most noticeable in portions of the missions when multiple robots are in a steady phase and trying to remain stationary with static relative positions: in these steady phases, much of the observed difference in error between simulation and reality can be attributed to the noise present in the aerial robot's hovering behavior, especially when the robot it is referencing for relative positioning is also trying to remain in place but has noise present. This is due to the fact that simulation models of the robots have been tuned according to the noise observed in sensors of a single robot, not the noise observed in interactive multi-robot behaviors.

For the interactions between the robots and their environment, objects on the ground (such as obstacles to be avoided, see Fig 3.10c,d) are also simulated using ARGoS's 3D-dynamics engine based on the ODE library, so that the ground robots can physically interact with them.

Regarding networking, in the simulations robots are allowed to send messages to other robots in their FoV every 0.2 s, which approximates the message rate in experiments with real robots, because the real robots have a step interval of 0.2 s according to their own internal clock.

(a) (b)

Figure 3.11: The (a) real and (b) simulated arenas. Note that the blue circles around robots in (b) are for visualization purpose only and are not part of the robot models. Without these blue circles, the dimensions of the robots proportional to the arenas in the real and simulated environments can be seen to be equivalent. The flight zone, outlined in black in (a) and outlined in blue in (b), is 6 m × 10 m with a height of 3 m above the floor. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

For the different experiment setups, customized environments in ARGoS are created using a python script to generate the .argos setup file. This generation can include arena walls, environmental features composed of obstacles that occur at certain positions relative to the size of the arena, and uniformly random initial positions of robots and positions of obstacles. In simulated experiment setups that are supposed to match a real experiment exactly (see Fig. 3.11), the generated arena walls in ARGoS match the border dimensions of the real arena floor. In other simulated experiments, in which more robots are used than in any real experiments, the arena

size is generated to be large enough for the respective system size.

In all experiments, the time step in the simulator is equivalent to 0.2 seconds. Data logging is executed at each time step and records the global position and orientation of each robot.

Our simulated experiments were run either on the experimenter's local machine or on our in-house IRIDIA computing cluster. The two cluster nodes we used have the following hardware: 2 AMD EPYC 7452 (32-core, 2.35GHz, 128MB L3 cache), 512GB RAM. Regarding computation work that was assessed using CPU clock cycles, note that each simulated robot was run as a single-threaded program and analysis was done in terms of clock cycles per robot.

To assist with experiment management, we use an "experiment supervisor" software custom-developed for this study. The experiment supervisor software is used: 1) before the experiment, to confirm all robots are active, correctly configured, seen by the motion capture system, and in the correct state; 2) to send a signal to all robots to start the experiment at the same time; 3) to record experiment data during the experiment; and 4) to send a signal to all robots to end the experiment at the same time. During each experiment, the experiment supervisor software records the positions and orientations output from the motion capture system as well as all information output from the robots.

During the experiments (i.e., between the signals sent to start the experiment and end the experiment), the robots do not receive any centralized commands or global information. The robots are allowed to communicate only with each other, and only under certain conditions. Communication between the robots occurs over a wireless network, and two robots are only allowed to communicate with each other if they are 'connected' or if one of them is in the other's field of view. In our indoor arena, the robots communicate using a wireless LAN. Messages between robots are routed by the experiment supervisor software, which has access to the MAC address associated to each robot ID.

During an experiment, the experiment supervisor software records the messages passed between robots, for the purpose of data logging. Note that, during an experiment, the robots do not receive any commands or information originating from the

experiment supervisor software; robots only communicate with each other.

3.8 Experiment environment

We conduct our experiments with real robots in an indoor arena custom-built for this thesis inside a multi-use room (see Fig. 3.12). The arena consists of an unobstructed flight zone and floor area surrounded by a safety buffer and removable security netting, with mounting positions above the center of the flight zone and around the perimeter. The arena is mounted with visual cameras and an optical motion capture system used for recording experiment data. Note that information from the motion capture system is not accessible by the robots and is only used for data logging.

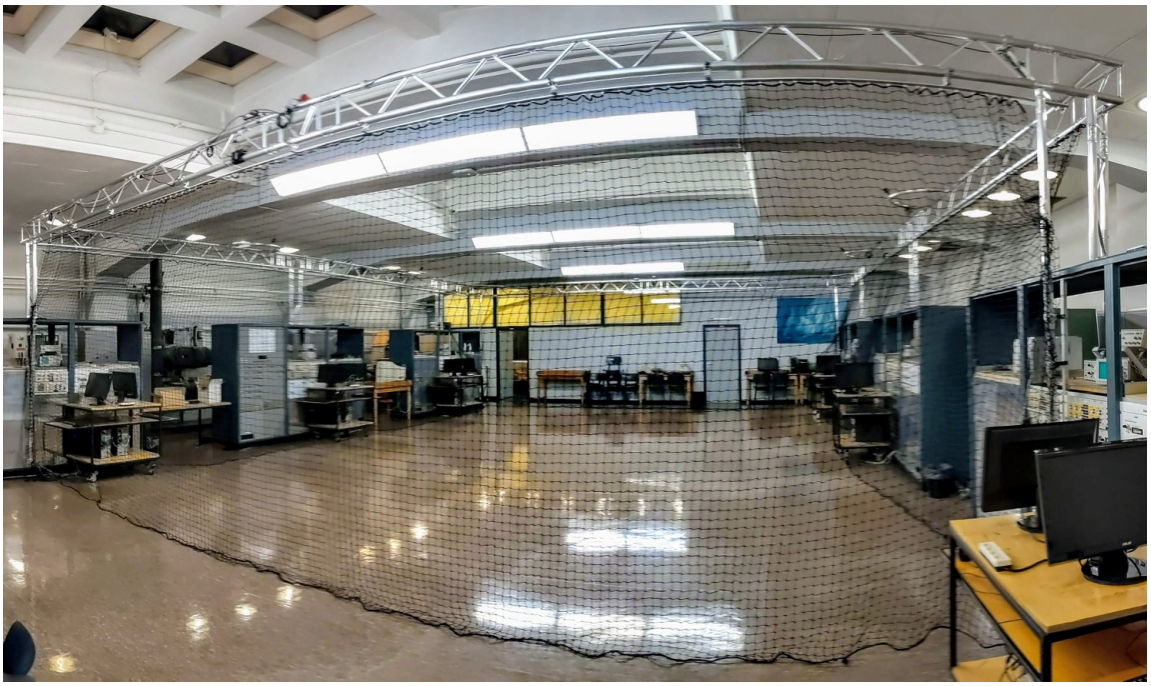


Figure 3.12: Wide-angle photograph of the indoor arena. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

The arena perimeter consists of a truss system built on top of permanent metal fixtures (see Fig. 3.13) from modular 2-point trussing segments (Global Truss F32 Ladder Truss family). This results in a rigid frame around the arena that allows

the visual and motion capture cameras to consistently record from the same fixed physical locations and orientations over experimentation periods of several months.

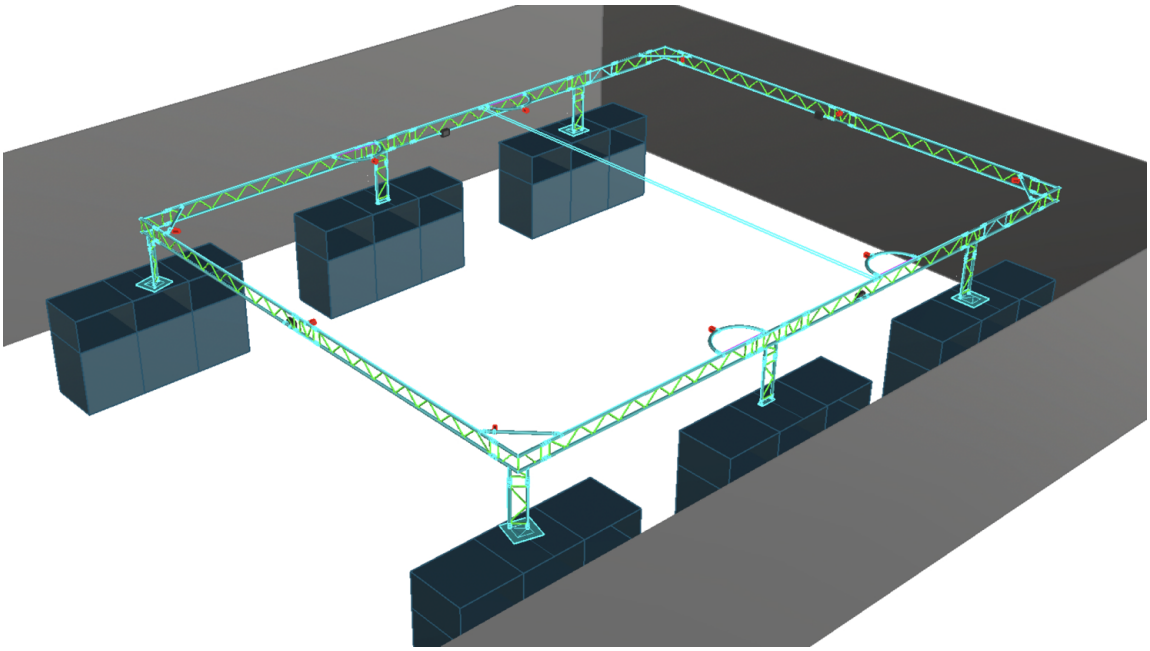


Figure 3.13: Perspective view of the truss system of the arena. There are 10 motion capture cameras (red) mounted on the truss around the perimeter of the arena. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

We use an infrared motion capture system with 10 cameras that each have a 56° field of view (OptiTrack Flex 13 family, 56° FOV, 5.5 mm, 800 nm long-pass IR, see Fig. 3.14).

The flight zone is 6 m × 10 m with a height of 3 m above the floor. The motion capture cameras are mounted in an 8 m × 12 m rectangle centered around the flight zone, at a height of 3.4 m above the floor. The cameras are mounted 4 m apart from each other (one in each corner, one in the center point of each of the shorter perimeter sides, and two at the one-third midway points of each of the longer perimeter sides, see cameras marked in red in Fig 3.13). Note that the truss system is larger than the rectangle of the camera mounting positions. With the cameras mounted in these positions, motion capture of the ground robots at the floor level and of the aerial

Figure 3.14: Motion capture cameras (red) at a side location (left) and corner location (right) in the arena. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

robots at the altitude of 1.5 m above the floor is reliable everywhere in the 6 m × 10 m flight zone, except in the corners. Therefore, the corners of the arena are cut at a 45° angle, 0.5 m away from the original corner point, to form the final octagonal arena shape used in the simulated and real experiments.

The 10 cameras are connected via cable to four USB hubs (OptiHub 2) that manage camera syncing. The hubs are each connected via cable to one PC station with motion capture software (OptiTrack Motive) that outputs the position and orientation data of the tracked rigid bodies.

In the experiments, we track each aerial or ground robot as a rigid body, using four passive markers (OptiTrack precision spheres with 3M 7610 reflective tape) affixed to the robot. In order for the robot positions and orientations to be tracked correctly, the four markers of one robot need to be mounted in a 2D planar configuration that is asymmetrical along both axes formed between a marker and its opposite (i.e., between two opposite vertices of the quadrilateral formed by the four markers). The marker configurations of the robots also need to be geometrically unique, including when they are rotated in 2D or 3D (i.e., each marker configuration must be unique in any rotated position). The unique marker configurations are provided by custom mounting plates made for this study (see example mounting plate designs

in Fig. 3.15) and are associated to the correct robot IDs using the motion capture software (OptiTrack Motive) at the beginning of each experiment session.

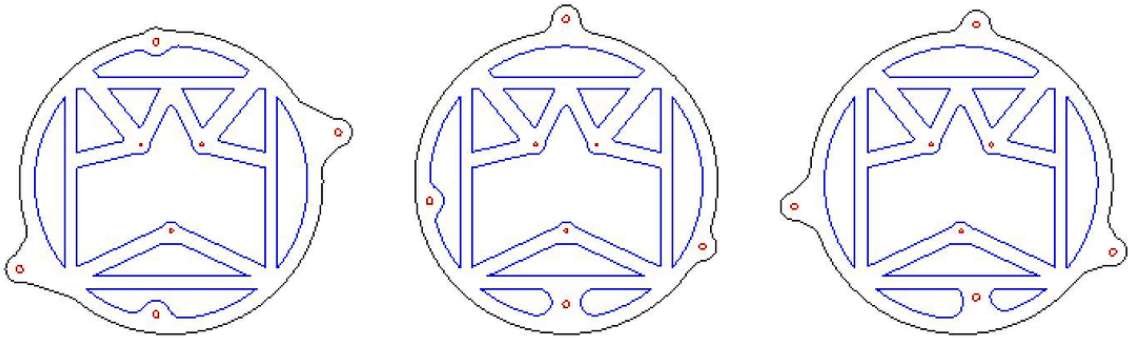


Figure 3.15: 2D mounting plate designs for unique and asymmetrical passive marker configurations for each robot, for use by the motion capture system. Three example designs for ground robots are shown. Large red circles near the outer perimeter of each plate indicate the mounting positions of the four passive markers. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

Besides recording the motion capture data, we also record experiment videos. We use two visual cameras: an HD action camera (GoPro) mounted to the ceiling, above the center of the arena, and a DSLR camera (Canon EOS 5D Mark IV) mounted to the truss system along the perimeter.

3.9 Demonstrations of the *S-drone*

In this section, we demonstrate the *S-drone* in basic flight operations and cooperative behaviors: single-UAV take-off and hover flight, single-UAV vision-based navigation, and two swarm demonstrations (one multi-UAV flight in a dynamic formation, and one multi-UAV object transport).

3.9.1 Take-off and hover flight

This experiment validates stable hover flight of the *S-drone* and reports deviation from the target position. In this setup, the *S-drone* tracks a static target position and hovers for 10 minutes after autonomous take-off. The take-off position is set as

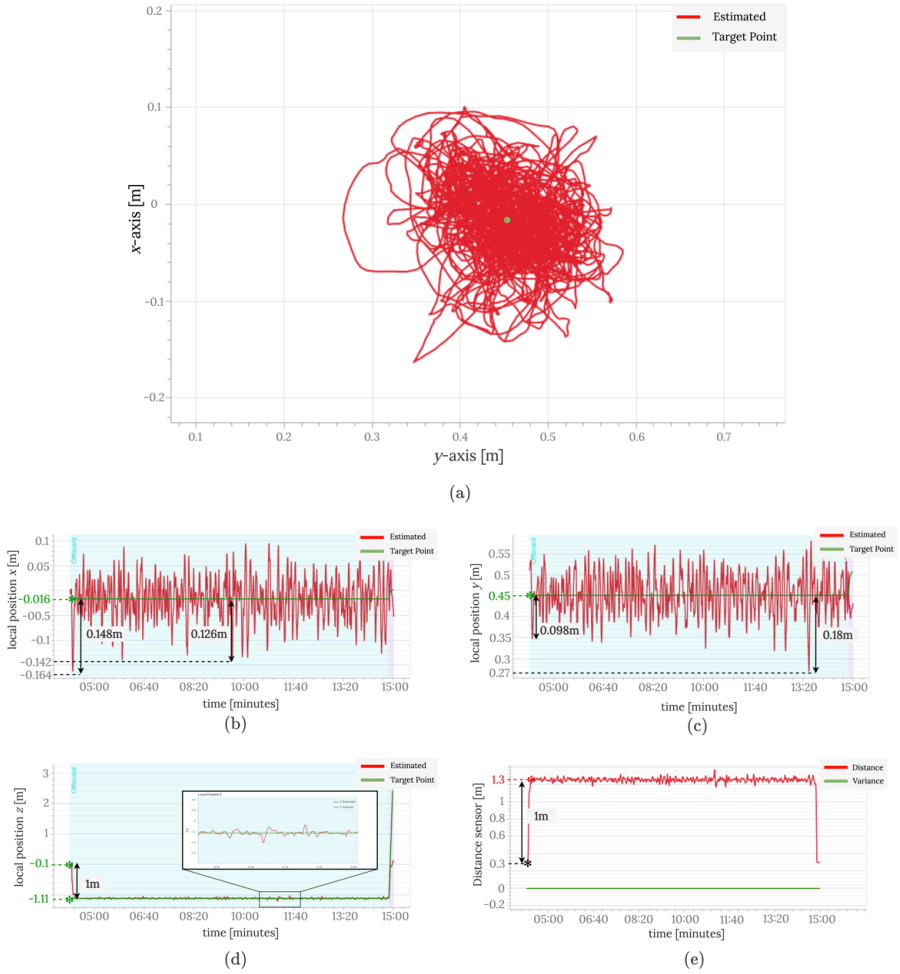


Figure 3.16: Autonomous indoor hover performance. (a) 10-minute flight data is visualized in the xy-plane. The UAV is commanded to hover at a fixed horizontal position. (b) Time evolution of the x position (red) relative to its constant reference value (green). (c) Time evolution of the y position (red) relative to its constant reference value (green). In both cases, the horizontal position remains within approximately ± 0.1 m of the target over the entire 10-minute experiment. (d) Relative z-coordinate values in the North-East-Down frame; the target altitude is constant and the maximum deviation is about 0.1 m. (e) Relative z-coordinate values read by the downward-facing distance sensor. (Reprinted from (Oğuz et al., 2024).)

the target position in the xy-plane, and the altitude target position is set to 1 m. Using the cascaded PID controller of the *S-drone*, the target position is fed to the position controller, which triggers the roll and pitch as control inputs.

Fig. 3.16(a) shows the xy-position of the *S-drone* during flight. The *S-drone* stays within a 0.2 m radius of the target xy-position, which is comparable to the performance of a human operator using a joystick to actuate pitch and roll. This deviation is reasonable for a UAV that uses onboard sensors for localization and does not use an external positioning system.

The x- and y-coordinates relative to the target position during flight are given in Figs. 3.16(b-c). For the x values, the maximum deviation from the target position (green line) during hover is 0.126 m and during take-off is 0.148 m. The mean deviation for the whole flight is approx. 0.13 m, with a standard deviation of 0.05 m. For the y-coordinate values, the maximum deviation from the target position during hover is 0.18 m and during take-off is 0.098 m. The mean deviation is approx. 0.16 m, with a standard deviation of 0.06 m. The slightly larger deviation on one axis compared to the other is because, in practice, the thrust of the rotors will never be perfectly balanced, and therefore some adjustment is required as the UAV gains altitude (i.e., it might pitch forward or backward).

The altitude (z-coordinate) values are given in Figs. 3.16(d-e). Fig. 3.16(d) shows the take-off altitude and the relative z-coordinate values during flight. Due to barometer and IMU sensor noise, the initial relative z-coordinate value during the experiment is 0.1 m. Therefore, the target position of 1 m above the initial take-off altitude is set at 1.1 m. The relative z-coordinate values of the UAV are around 1.1 m during the flight, with a maximum deviation of approx. 0.1 m and an average deviation of approx. 0.05 m, and a standard deviation of 0.03 m. Fig. 3.16(e) shows the altitude values measured by the downward-facing single-point LiDAR. The take-off altitude read from the LiDAR is 0.3 m (when the *S-drone* is sitting on the ground). Therefore, a LiDAR reading of 1.3 m during flight indicates a true relative altitude of 1 m. The LiDAR measurements fluctuate around 1.3 m with small variations (standard deviation 0.03 m), confirming that the *S-drone* maintains the desired 1 m hover altitude throughout the experiment.

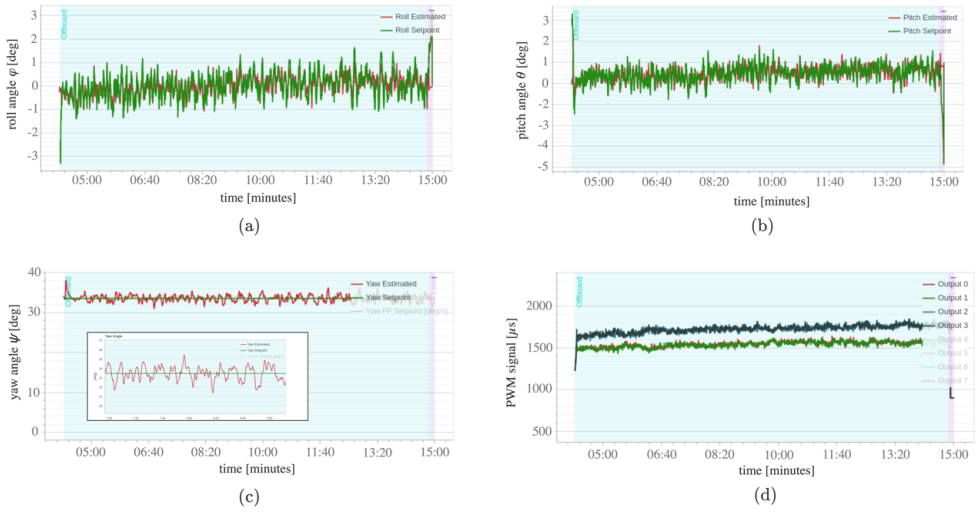
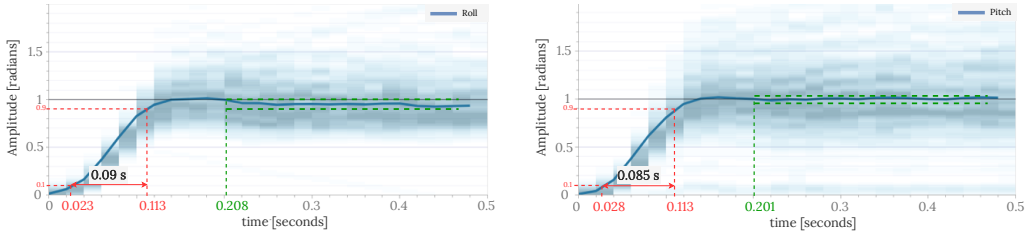


Figure 3.17: Autonomous indoor hover performance during a 10-minute flight. (a) Roll angle. (b) Pitch angle. (c) Yaw angle. (d) Control signals that are sent to the individual motors. (Reprinted from (Oğuz et al., 2024).)

Another important metric for UAVs is attitude control performance during hover. The attitude controllers must keep the attitude angles (i.e., roll, pitch, and yaw angles) in certain intervals during hover. In Figs. 3.17(a–c), the recorded roll, pitch, and yaw values of the *S-drone* remain close to their respective constant reference values. Roll and pitch stay within approximately $\pm 1.5^\circ$ around their references, and the yaw angle shows only small variations around a fixed heading. The standard deviations of the roll, pitch, and yaw errors are on the order of $\sigma_\phi \approx 0.5^\circ$, $\sigma_\theta \approx 0.6^\circ$, and $\sigma_\psi \approx 0.7^\circ$, respectively. These oscillations are small enough that they are not visually observable during flight, indicating stable attitude control.

The attitude control performance of a UAV is directly linked to the motor control signals. Fig. 3.17(d) reports the signals sent to the individual motors of the *S-drone* over the 10-minute hover experiment. These values remain between the configured minimum and maximum PWM limits (here, from 1 000 to 2 000 µs) and do not exhibit excessive noise. Fig. 3.17(d) also reflects the mass distribution of the specific *S-drone* used in this experiment (a second, separately assembled *S-drone* would show

slightly different traces). Two motors (labelled Output 2 and Output 3) operate with somewhat higher average thrust than the other two, indicating a slight mass imbalance introduced during assembly. This imbalance may reduce the available thrust margin and put more load on those motors; however, the onboard flight controller compensates for it so that overall flight stability is not affected.



(a) (b)

Figure 3.18: Unit step input responses of the *S-drone* controllers. The red arrows indicate the time interval required for the response to rise from 10% to 90% of its final value. The vertical green lines show the settling time, which is the time it takes for the response to be stably within a certain range of the final value (horizontal green lines). (a) The step response of the roll rate controller. (b) The step response of the pitch rate controller. (Reprinted from (Oğuz et al., 2024).)

Finally, another metric for UAVs is the unit step input response of the controllers. The step response is important because large and rapid deviations from the long-term steady state can potentially have extreme effects on a UAV. Formally, the step response of a controller provides information about the overall stability of the vehicle and its ability to reach a steady state. From a practical point of view, it is important to know how a UAV responds to sudden input changes. Fig. 3.18 shows the step response of the roll and pitch rate controllers of the *S-drone* with rising and settling times. The rise time of both the roll and pitch rate responses is approx. 0.09 s and the settling time is approx. 0.2 s without overshoot. In this demonstration, the responses of the *S-drone* were fast enough to maintain stability and the steady state errors remained relatively small.

3.9.2 Vision-based navigation

This experiment validates autonomous flight and vision-based navigation with the *S-drone* in a GPS-denied environment. In this setup, the *S-drone* autonomously takes-off, hovers, and flies with a velocity of 0.2 m/s while performing waypoint navigation by guiding itself with fiducial markers (tags).

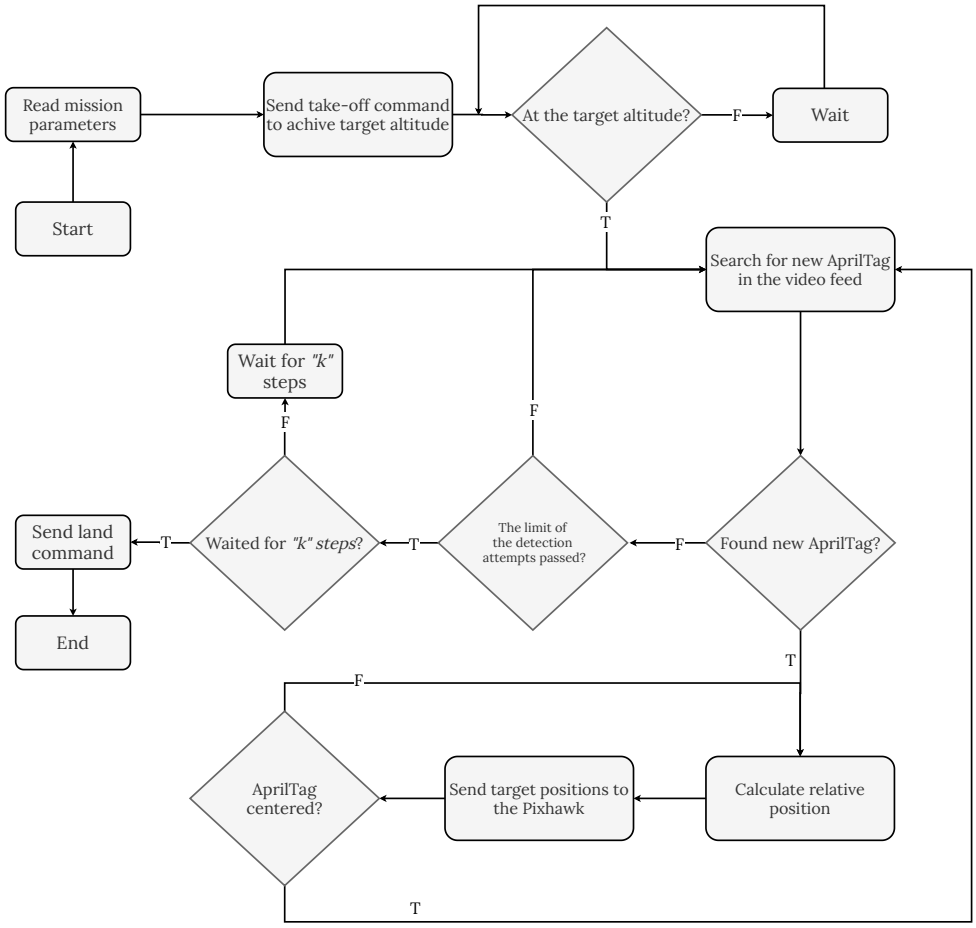


Figure 3.19: Control algorithm flow chart for vision-based navigation with the *S-drone*. (Reprinted from (Oğuz et al., 2024).)

The flow chart of the control algorithm used for vision-based navigation is given in Fig. 3.19. After it receives a command to start and take-off, the *S-drone* flies upward until it reaches its target altitude and then begins searching for tags in order

to start navigating the environment. When it finds a tag, the *S-drone* processes the frame, retrieves the tag information, estimates the position and orientation of the tag relative to itself, and prepares to fly towards the tag. It calculates a two-dimensional vector for its new target position (i.e., the tag position) relative to its current position. The target position is then provided to the Pixhawk4 flight controller and the *S-drone* moves towards the detected tag and hovers above its center (within a specified offset). After the first waypoint is achieved, the *S-drone* looks for new tags. If the *S-drone* detects a new tag, it repeats the process described above. If no tag is detected at first, the *S-drone* waits for a certain amount of time and makes another detection attempt while hovering and holding its current position. After a certain number k of unsuccessful detection attempts, it decides to land.

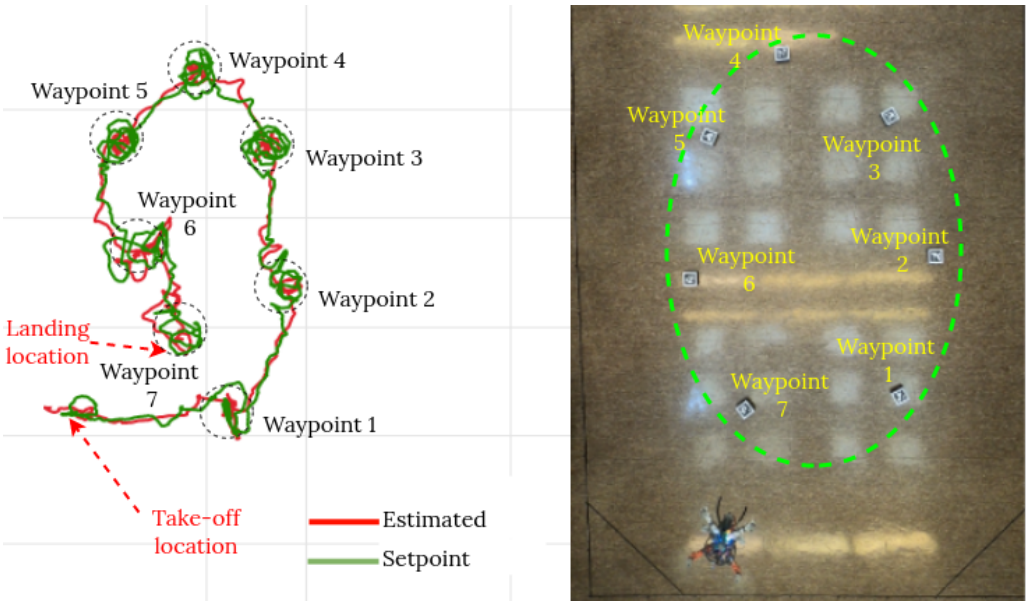


Figure 3.20: Autonomous vision-based waypoint navigation. (Reprinted from (Oğuz et al., 2024).)

In the experiment shown in Fig. 3.20, the *S-drone* uses this control algorithm to follow an ellipse-like path composed of seven waypoints. Fig. 3.20 shows the setup of this vision-based navigation experiment and reports the xy positions of the *S-drone* during flight.

3.9.3 Swarming capabilities

We conduct two multi-UAV demonstrations. The first is **Flying in a formation**, in which three *S-drones* fly together in a formation, while updating and rotating the shape, forming a series of target formations during flight. To establish, maintain, and reconfigure these formations, each *S-drone* needs to determine the current differences between (a) the real relative distances and relative orientations of neighboring *S-drones* and (b) the current targets. This inter-robot tracking is accomplished using our method for cooperative feature-level sensor fusion between robots. Using this inter-robot tracking, each *S-drone* determines its relative spatial relationship with neighboring *S-drones* and attempts to reach its targets. This first demonstration (see Fig. 3.21 and the online movie [6]) shows the following procedure: **(1) Take-off:** Each of the three UAVs takes off, stabilizing near a target altitude of 1.5 m. **(2) Triangle Formation:** After take-off, the UAVs establish a triangular formation. **(3) Formation Rotation:** With the triangular formation shape intact, the UAVs undertake a synchronized turn. **(4) Line Formation:** The UAVs then reconfigure into a linear arrangement according to new targets for their relative positions and orientations.

The second demonstration is **Cooperative object transport**, in which two *S-drones* collaborate to transport a hung payload (i.e., an object hanging from two attachments, one to each *S-drone*). Here, the *S-drones* again use inter-robot tracking via cooperative feature-level sensor fusion to maintain their target relative spatial relationship, in this case while collaboratively lifting and transporting a hung payload. This demonstration (see Fig. 3.22) shows the following procedure: **(1) Take-off:** With the payload securely attached, the UAVs execute a coordinated take off, keeping the payload supported on both sides, and stabilizing near a target altitude of 1.5 m. **(2) Object Transport and Formation Maintenance:** After take-off, the UAVs move towards a target landmark while maintaining a side-by-side formation with fixed target UAV–UAV x, y distances. The UAVs need to continuously adjust to each other's movements in order to maintain this formation despite disturbances

[6] https://osf.io/h9azb

introduced because of the hung payload. **(3) Endpoint Descent:** When reaching the target landmark, the UAVs execute a coordinated landing to achieve gentle placement of the payload.

3.10 Chapter conclusions

In this chapter, we have presented the *S-drone*, a completely open-source UAV platform designed to support the testing and validation of the algorithms and methods proposed in this thesis. The *S-drone* represents a significant step forward in enabling practical swarm robotics research with UAVs, particularly by leveraging onboard sensors and positioning systems to achieve autonomous operation without reliance on external infrastructure.

We outlined the challenges associated with UAV swarm robotics, emphasizing the need for platforms that are modular, extensible, and cost-effective. In response, we designed the *S-drone* with a flexible hardware and software architecture to enable rapid prototyping and seamless deployment of swarm algorithms. To complement this, we provided the software framework for simulating and validating swarm behaviors, effectively bridging the gap between theoretical models and real-world implementations.

Figure 3.21: Demonstration of multi-UAV flight in a dynamic formation: (a) Initial random placement of the three UAVs. (b) Take-off procedure. (c) Establishment of a triangle formation. (d) Alternative perspective of the established triangle formation. (e) Start of formation rotation. (f) UAVs in rotation. (g) Completion of the rotation. (h) Final reconfiguration transitioning from triangular to linear formation. (Reprinted from (Oğuz et al., 2024).)

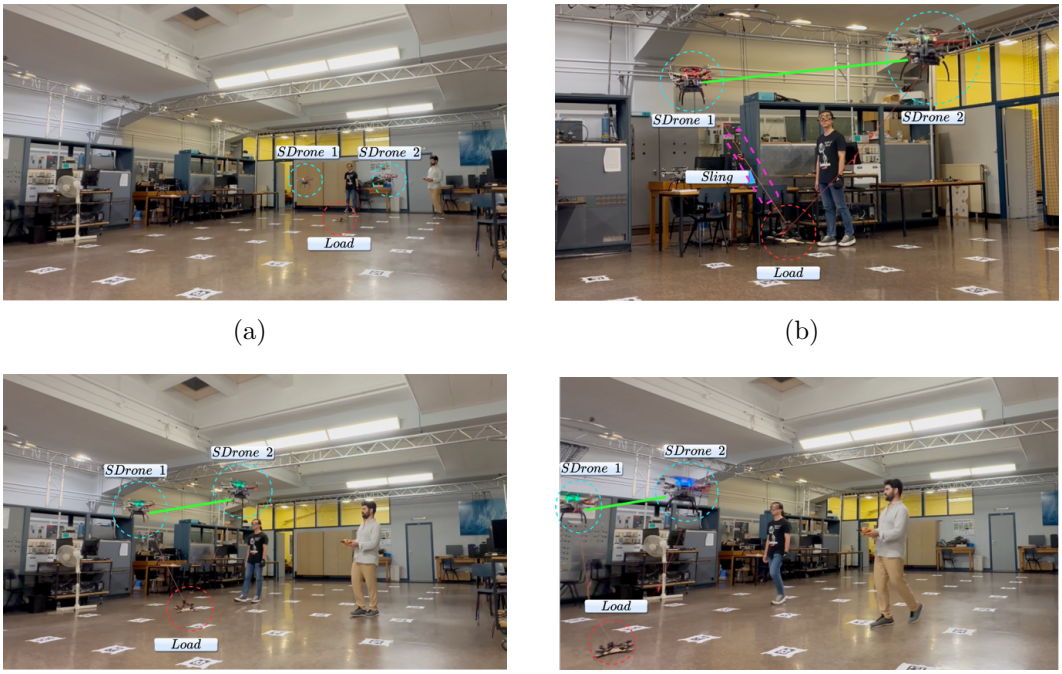


Figure 3.22: Demonstration of multi-UAV object transport: (a) Coordinated take-off. (b) UAVs navigating the environment. (c) Arrival at the target destination. (d) Coordinated descent ensuring safe payload placement. (Reprinted from (Oğuz et al., 2024).)

Chapter 4

Self-organizing Hierarchy for Robot Swarms

This chapter[1] *presents the Self-organizing Nervous System (SoNS), a robot swarm architecture based on self-organized hierarchy. The SoNS approach enables robots to autonomously establish, maintain, and reconfigure dynamic multi-level system architectures. For example, a robot swarm consisting of n independent robots could transform into a single n-robot SoNS and then into several independent smaller SoNSs, where each SoNS uses a temporary and dynamic hierarchy. Leveraging the SoNS approach, we show that sensing, actuation, and decision making can be coordinated in a locally centralized way, without sacrificing the benefits of scalability, flexibility, and fault tolerance. We demonstrate that the capabilities of the SoNS approach greatly advance the state of the art in swarm robotics. The missions are conducted with a real heterogeneous aerial–ground robot swarm, using the robotic platforms introduced in the previous chapter. We also demonstrate the scalability of the SoNS approach in swarms of up to 250 robots in a physics-based simulator, and demonstrate several types of system fault tolerance in simulation and reality.*

The experiments in this chapter make use of the hardware, software, simulation environment, and support software for managing experiments that were presented in Chapter 3.

[1]The content of this chapter was previously published in (Zhu et al., 2024); the results presented in Section 4.3 were included in the supplementary materials of (Zhu et al., 2024).

4.1 Approach and contributions

In the SoNS concept, robots self-organize into dynamic multi-level system architectures using ad-hoc remote bidirectional connections. The result is a swarm composed of a number of reconfigurable Self-organizing Nervous Systems—SoNSs, see Fig. 4.1. In each SoNS instance, each "child" robot has chosen to temporarily grant explicit supervisory powers to a "parent" robot in the level above it, culminating in a single "brain" robot that acts as a temporary coordinating entity. The structure of a SoNS instance is defined both by the topology of its connections and the relative robot poses associated with those connections. Thus, the underlying graph of a SoNS is a rooted tree with bidirectional edges and a set of attributes.

One of the aims of the SoNS concept is that the key benefits of self-organization should be maintained in the SoNS architecture. The first key benefit that should be present is scalability. For instance, a new connection should not become any more difficult to establish if a SoNS has more members. For this reason, each connection in a SoNS is managed solely by the two robots sharing the connection, and each robot only communicates with robots in its field of view. In other words, unlike in a centralized system with a root node, the brain robot does not communicate with all its members directly. The second key benefit that should be present is fault tolerance. For this reason, every robot in a SoNS is replaceable—even the brain. If any robot fails or is lost, another robot in its SoNS or recruited from its surroundings can replace it automatically. If no extra robots are available, a SoNS can reconfigure using the robots it already has and try to continue with its mission.

Another aim of the SoNS concept is that the members of one SoNS should be able to act seamlessly as a single entity despite only communicating with nearby robots. For this reason, at each connection in a SoNS, the child robot sends sensor information upstream and the parent robot sends actuation instructions downstream, allowing the brain to steer reactions of the SoNS as a whole. However, to be able to react in an agile way to its environment and task requirements, the actions of robots in a SoNS should not be strictly hierarchical. The SoNS should be able to adjust its inter-level control distribution—that is, the effective degree of centralization or

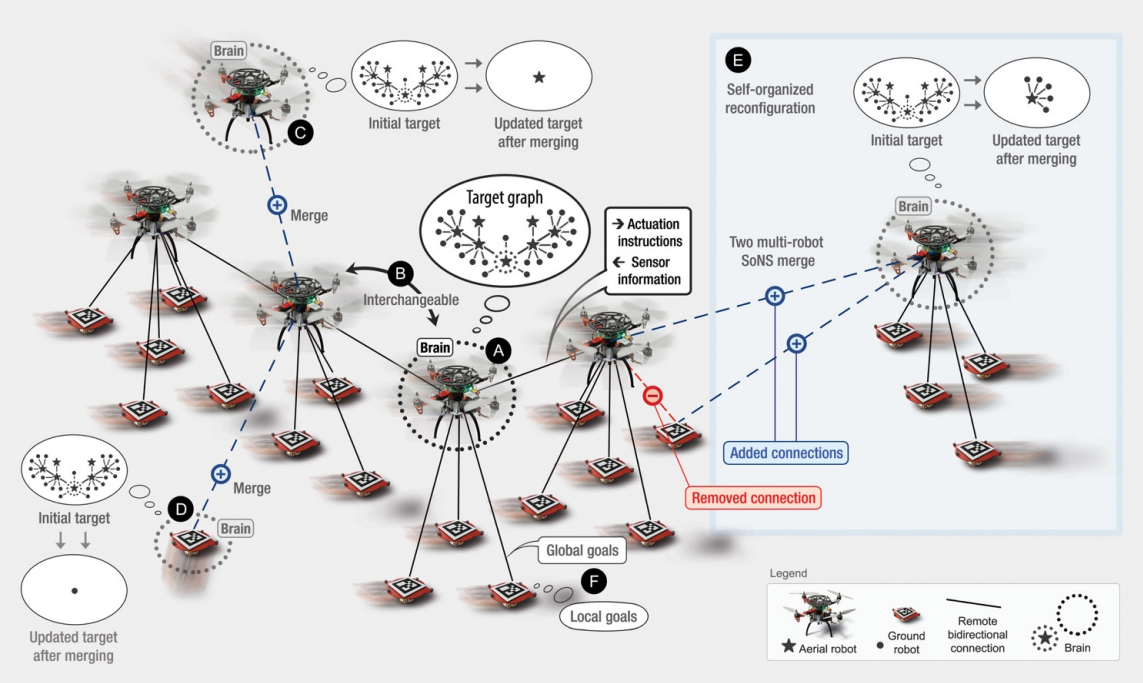


Figure 4.1: The Self-organizing Nervous System (SoNS) concept. Robots self-organize reconfigurable multi-level system architectures using communication only with nearby robots. (A) In a SoNS, each "child" robot has chosen to temporarily grant explicit supervisory powers to a "parent" robot in the level above it, culminating in a single "brain" robot that acts as a temporary coordinating entity. At each bidirectional connection, the child robot sends sensor information upstream and the parent robot sends actuation instructions downstream. (B) Any robot at any level of hierarchy can be interchanged with another robot—even the brain. (C,D) At initialization, each robot is the brain of its own single-robot SoNS and has a target graph. If it encounters another SoNS, it can choose to accept recruitment and merge with it, thereby abdicating its "brain" status. (E) The process by which robots establish and maintain SoNS connections is self-organized and SoNS topologies can be reconfigured by reallocating robots (i.e., removing and adding connections within the same SoNS), merging two or more SoNS, or splitting a SoNS. Here, we see a five-robot SoNS (right) that merges with a 20-robot SoNS (left). (F) The SoNS topology is used to organize supervisory powers and send actuation instructions downstream, but child robots are still semi-autonomous and can adjust the inter-level control distribution in the SoNS, adapting the effective degree of centralization or decentralization in the decision-making processes of the SoNS. (Reprinted from (Zhu et al., 2024).)

decentralization of decision making. Therefore, the inter-level control distribution in a SoNS is not strictly determined by the connection topology. Rather, any robot in a SoNS can set its own local actuation goals (e.g., avoid a small obstacle) as well as "global" (SoNS-wide) actuation goals that will be sent both upstream and downstream (e.g., for avoiding a mobile source of danger in an emergency). Also, when a robot in a SoNS combines its local goals with instructions received from its parent and any global (SoNS-wide) goals, it can adjust the weighting according to its available sensor information.

The final aim of the SoNS concept is that a user should be able to program the whole robot swarm as if it were a single robot with a reconfigurable morphology. For this reason, the system architecture and behavior of a SoNS can be reorganized by the brain on demand (see Fig. 4.1). Thus, in a single program that runs on every robot, the user can directly code not only actions to be taken by the brain, but also actions to be taken by the whole SoNS. This can include changes to the target topology, target relative poses, target motion trajectories, and other actuation goals. In principle, the user should also be able to program multi-SoNS behaviors as if they were multi-robot behaviors.

4.2 SoNS control algorithm

At initialization, each robot is an independent single-robot SoNS, of which it is the brain by default. A multi-robot SoNS is established by merging two or more SoNSs (initially, two single-robot SoNSs). Each robot, when it initializes as a brain, starts with mission-specific goals and the target graph of the SoNS that it would like to build downstream from it, running a local copy of the SoNS control algorithm (see Fig. 4.2). As a robot attempts to fulfill its goals and complete its target graph, it will participate in self-organized multi-robot behaviors by receiving inputs from and sending outputs to robots in its field of view. The key components of the SoNS control algorithm (see Fig. 4.2) running on each robot are: A) update target graphs, B) update node attributes, C) manage connections to neighbors, D) transform input vectors (i.e., reference vectors received from neighbors) to own coordinate frame,

E) classify sensor information, and F) update actuation instructions. Using these key algorithm components robots self-organize the following multi-robot behaviors: merging, node allocation, splitting, collective sensing, and collective actuation.

Merging. Every robot, whether it is a brain or the child of another robot, tries to recruit any robot that appears in its field of view (FoV). When two robots appear in each other's FoV, they send each other messages to compete for recruitment. When a robot receives a recruitment message, it rejects the recruitment if it is or recently was in the same SoNS as the competitor, or if the quality (which by default is randomly generated but can be redefined in mission-specific ways) of the competitor's SoNS is lower than its own; otherwise, it accepts. If a robot accepts recruitment, it becomes the child of its competitor, splitting from its previous parent if it had one; if it receives an acceptance, it becomes the parent of its competitor. When a recruitment is accepted, the new child and all its downstream robots (if it has any) will merge into the SoNS of the new parent. For a recruited child to be assigned to a node in its parent's target graph, it needs to undergo node allocation.

Node allocation. Each robot has a graph describing the targets of the robots downstream from it, but not a graph of their current configuration. The current configuration can differ from the target graph in terms of topology or graph attributes, which include relative poses. For the configuration of a SoNS or SoNS branch to approach the target graph associated with it, robots in a SoNS need to self-organize their node allocation operations in the following manner. During node allocation, any robot with at least one child can either assign its children to nodes in its target graph or pass them upstream to be handled by its parent. To make these decisions, the robot tries to pair its children with nodes in its and its parent's target graphs, based on the current and target relative poses and downstream vertex cardinalities. All children, including those already assigned to a node in the robot's target graph, can be (re)allocated at every time step. Thus, a parent can replace one of its already assigned children with a new unassigned one that is currently a better match for the target node, effectively demoting the formerly assigned child to unassigned status until the next allocation iteration. Because of this replacement possibility, the robots in a SoNS can redistribute themselves continuously within the structure of

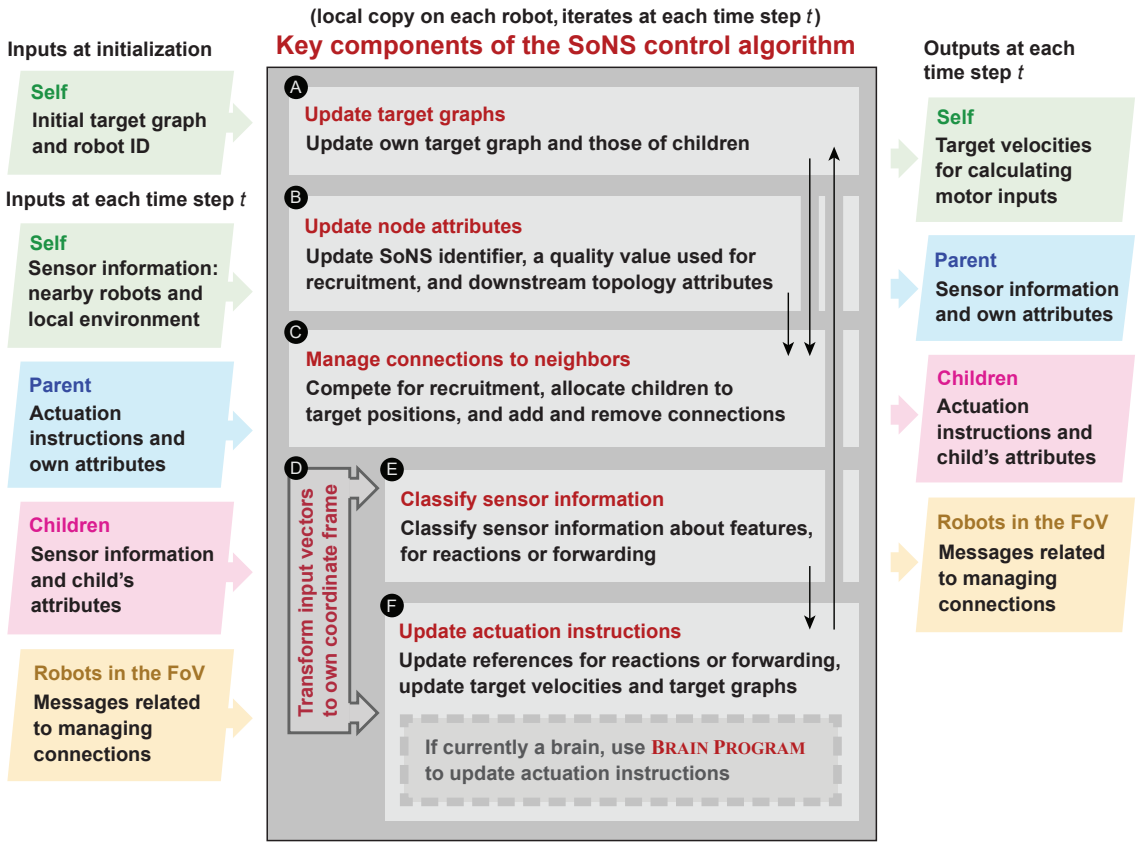


Figure 4.2: The key components of the SoNS control algorithm. Each robot (A) updates its target graph (i.e., the target connection topology and target relative poses) and those of its children, (B) updates its node attributes, (C) manages its connections to neighbors, (D) transforms its input vectors into its own coordinate frame, (E) classifies sensor information, and (F) updates its actuation instructions. These components receive inputs from and send outputs to other components of the SoNS algorithm (black arrows between components), the robot's other software layers (green), parent (blue) and/or children (pink), and other robots appearing in its field of view (FoV) (yellow). (Reprinted from (Zhu et al., 2024).)

their SoNS.

Collective sensing. A robot in a SoNS can directly react to sensor information it gathers itself or receives from any children, using feature-level representations (e.g., symbolic representations of detected object types and their relative poses). Each robot that has a parent classifies features it has detected or received as either features to react to or features to forward to its parent. Features that it decides to react to are classified as either requiring standard (local) reactions or emergency (SoNS-wide) reactions. If a robot has no parent (i.e., is a brain), it inputs the sensor information it gathers or receives to a mission-specific BrainProgram.

Collective actuation. When a robot is a brain, its mission-specific BrainProgram becomes active. Using this module, a brain defines and follows its own actuation instructions (i.e., its target linear and angular velocities and its target graph) independently of the robots downstream from it, except in the case of global reactions. When a robot has a parent, it acts as a semi-autonomous follower, taking into account three types of reference vectors. The first type is local, based on environment features from its own sensor information; the second is hierarchical, based on the target relative pose the robot receives from its parent; and the third is global, based on any environment features that require a SoNS-wide reaction, such as emergencies. Global references can be received from any directly connected robot (parent or child) or can be generated based on features from the robot's own sensor information. Each robot forwards any global references it generates or receives to every robot it is directly connected to, except for the robot that sent the respective reference. Using these reference vectors (local, hierarchical, global), each child robot then defines its own target linear and target angular velocities.

Splitting SoNSs. A robot in a SoNS can expel a child, for instance based on a command received from its parent. If it chooses to expel a child, or if the child is unintentionally disconnected, the SoNS splits into two. The expelled child will automatically resume being the brain of its own SoNS and will update its target graph accordingly. If the expelled robot has any children, all its downstream connections are maintained. The downstream connections can then be reorganized in the next time steps, using the robots' updated target graphs and node allocation iterations.

The primary contribution of the SoNS robot swarm architecture is that a robot system can integrate the manageability advantages of centralized systems without sacrificing the scalability, flexibility, and fault-tolerance advantages of self-organized systems. This contribution is founded on the novel combination of four impactful features for robot swarms (see Fig. 4.1): self-organized controllable hierarchy, interchangeable leadership (i.e, interchangeability of the brain), explicit inter-system reconfiguration, and reconfigurable swarm behavior structures. Together, these features enable robot swarms to self-organize reconfigurable multi-level system architectures, including their communication structures, control distributions, and system behaviors.

Self-organized controllable hierarchy. The SoNS approach allows a robot swarm to self-organize a temporary hierarchical communication structure that (i) is built and maintained using communication only with nearby robots, (ii) is not imposed from the outside, and (iii) is comprehensively controllable (that is, the SoNS-wide multi-level structure can move from any initial state to any desired state in its configuration space of directed acyclic graphs). In other words, a SoNS hierarchy can be explicitly defined and redefined by the brain and the desired changes occur through robots configuring and reconfiguring their local connections in a self-organized way. Self-organized controllable hierarchy has been shown in physically-connected robots but is novel in robot swarms, which so far have shown only emergent hierarchy (for example, (Firat et al., 2020)), not controllable hierarchy. The remote connections of robot swarms bring significantly different requirements than physical connections. The physical locations and topology constrain each other less under remote connections than under physical connections, which provides much more flexibility in how a SoNS can be organized, but also adds the challenge that physical location and topology must both be actively (and sometimes separately) maintained.

Interchangeable leadership. In a SoNS, all robots occupy an explicitly defined position in a hierarchy, but any robot, at any level of hierarchy, can be interchanged autonomously and on demand. This interchange is self-organized by the robots, using communication only with nearby robots. This means that, if the brain fails, the SoNS self-organizes to automatically and immediately substitute it with the nearest

robot, which continues to specify the same SoNS structure and mission goals as the previous leader. This is a novel feature for robot swarms that base their behaviors on communication with nearby robots and yet contain explicit leaders, which so far have used static (and sometimes manually defined) leadership. It is also a departure from many other types of multi-robot systems, which often use indiscriminate followers or groups of followers, but have not yet developed approaches for an explicit control hierarchy that is self-organized using communication only with nearby robots.

Explicit inter-system reconfiguration. The SoNS approach allows reconfiguration between multiple SoNS—that is, several SoNSs can split and merge themselves in a self-organized way that is coordinated by the brains of the SoNSs and uses communication only with nearby robots, without losing the existing sub-structures that could be retained. For example, several independent SoNSs could agree to merge simultaneously, and the robots would reorganize themselves around the new shared brain, retaining sub-sections of the old structures when possible. Robot systems in the literature have shown splitting and merging of sub-swarms or sub-teams, but with the non-leader members being unranked (for example, (Ducatelle et al., 2009, 2011)), so no reconfiguration of explicit sub-system architectures was demonstrated.

Reconfigurable swarm behavior structures. The inter-level control distribution and system behaviors within a SoNS (for example, the global structure defining which information sources influence which actions) can be negotiated and explicitly reconfigured without breaking the system architecture. Reconfiguration can occur (i) locally and temporarily to balance conflicting global and local goals; (ii) SoNS-wide for the purpose of global sensing, actuation, and decision-making goals set by the brain; but also (iii) locally for internal reorganization of a SoNS (for example, robots automatically redistributing themselves to compensate for a failed robot). Based on this capacity for internal reorganization, if the needed changes to behavior are too substantial to be managed by inter-level negotiation, an entirely new SoNS architecture can be initialized by the brain and self-organized by the robots. No existing work has presented a robot swarm architecture with these explicit reconfiguration capabilities.

4.2.1 Details

Each robot in a SoNS runs a local copy of the SoNS control algorithm which consists of key components for updating target graphs, updating node attributes, managing connections to neighbors, transforming input vectors to own coordinate system, classifying sensor information, and updating actuation instructions.

Within the SoNS control algorithm, there is one module that has no default definition and therefore is always mission-specific: the BrainProgram. This module serves as the primary programming interface, allowing a user to program the whole SoNS as if it were a single robot with a reconfigurable morphology. The other key functions used in the SoNS control algorithm have default definitions but can be treated as parameters depending on the robot platforms utilized and/or task requirements.

A visual overview of the SoNS control algorithm running on each robot is provided in Fig. 4.3. Definitions of the referenced variables and message types are listed in Tables 4.1 and 4.2. Key functions are presented both as equations (see Eqs. F1-F9 in Table 4.3) and as pseudocode (see Algs. 1-5). Each version of the BrainProgram is detailed in Alg. 6, while the parameterized function is provided in multiple versions in Alg. 7.

4.2.2 Functions of the SoNS algortihm

The key functions of the SoNS control algorithm are described below.

Target graphs: Each robot r_i updates its target graph and mission status either based on information received from its parent or, if it is a brain, according to its own information. It then updates its target downstream vertex cardinality $\text{VertexCard}^*_{r_i}$ using the function TargetCard (Eq. F1) on its target graph $\boldsymbol{H}^*_{r_i}$. If the robot has children, it extracts the target graph for each child $\boldsymbol{H}^*_{c_{r_i}}$ from its own target graph using the function Subgraph (Eq. F2) and sends it to the respective child.

Node attributes: Each robot estimates its actual downstream vertex cardinality $\text{VertexCard}^*_{r_i}$ and vertex height $\text{VertexHigh}^*_{r_i}$ using the functions SumCard

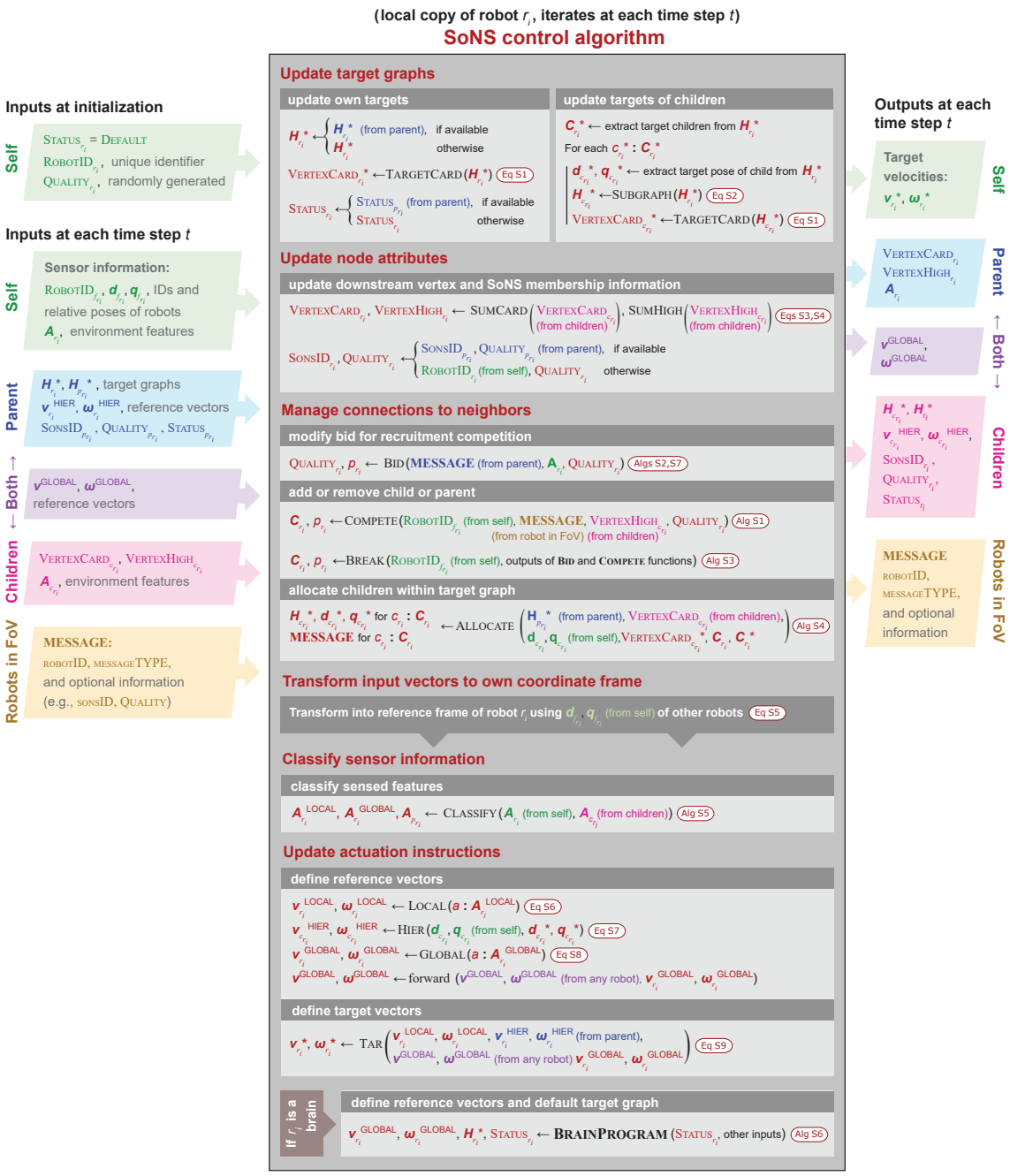


Figure 4.3: The SoNS control algorithm. Variables are colored according to their source: from the robot's own SoNS algorithm (red), from the robot itself but from outside the SoNS algorithm (green), from the parent (blue), from a child (pink), from both parent and children (purple), or from a robot in the field of view (yellow). The BrainProgram is the primary software module by which the user can effectively program the whole SoNS as if it were a single robot with a reconfigurable morphology. It is programmed for each mission separately. (Reprinted from (Zhu et al., 2024).)

Variable	Description
r_i	The robot on which the local copy of the algorithm is running
$\textsc{RobotID}_{r_i}$	Arbitrary unique identifier of robot r_i
$\textsc{Quality}_{r_i}$	SoNS quality of robot r_i
$\textsc{SonsID}_{r_i}$	SoNS identifier of robot r_i
$\textsc{SonsID}_{r_i}^{\text{OLD}}$	SoNS identifier of the most recent former SoNS of robot r_i
$t_{r_i}^{\text{OLD}}$	Estimated time steps since leaving the most recent former SoNS of robot r_i
$\textsc{VertexCard}_{r_i}$	Downstream vertex cardinality of robot r_i
$\textsc{VertexHigh}_{r_i}$	Vertex height of robot r_i
$\boldsymbol{H}_{r_i}^*$	Target graph that robot r_i is currently attempting to have built downstream from it
$\textsc{Status}$	The status that robot r_i will use if it is or becomes a brain
$c_{r_i} \in \boldsymbol{C}_{r_i}$	Child of robot r_i
p_{r_i}	Parent of robot r_i
$f_{r_i} \in \boldsymbol{F}_{r_i}$	Robot in the FoV of robot r_i
$\boldsymbol{A}_{r_i}$	Environment features detected by robot r_i (directly, or via any children)
$\boldsymbol{d}_{c_{r_i}}^*, \boldsymbol{q}_{c_{r_i}}^*$	Target displacement and relative orientation of a child of robot r_i
$\boldsymbol{d}_{f_{r_i}}, \boldsymbol{q}_{f_{r_i}}$	Sensed displacement and relative orientation of a robot in the FoV of robot r_i
$\boldsymbol{v}_{r_i}^*, \boldsymbol{\omega}_{r_i}^*$	Target linear velocity and target angular velocity of robot r_i
$\boldsymbol{v}^{\text{LOCAL}}, \boldsymbol{\omega}^{\text{LOCAL}}$ $\boldsymbol{v}^{\text{HIER}}, \boldsymbol{\omega}^{\text{HIER}}$ $\boldsymbol{v}^{\text{GLOBAL}}, \boldsymbol{\omega}^{\text{GLOBAL}}$	Reference vectors, see Eqs. F6-F8 in Table 4.3

Table 4.1: Key variables used in the SoNS control algorithm. (Reprinted from (Zhu et al., 2024).)

Message type	Recipient type	Message contents
all	—	RobotID of sender, RobotID of target recipient, message type, optional instruction to modify any parameter(s)
Recruit	f_{r_i}	SonsID and Quality of sender
RejectRecruitment	f_{r_i}	—
AcceptRecruitment	f_{r_i}	—
Split	c_{r_i}	destination node in target graph $\boldsymbol{H}_{c_{r_i}}^*$
HandOver	c_{r_i}	RobotID of target parent, $\boldsymbol{v}^{\text{HIER}}, \boldsymbol{\omega}^{\text{HIER}}$ towards target parent
UpdateParent	p_{r_i}	$\textsc{VertexHigh}_{r_i}$, $\textsc{VertexCard}_{r_i}$, a subset of $\boldsymbol{A}_{r_i}$, and optionally $\boldsymbol{v}^{\text{GLOBAL}}, \boldsymbol{\omega}^{\text{GLOBAL}}$
UpdateChild	c_{r_i}	$\textsc{Quality}_{r_i}$, $\textsc{SonsID}_{r_i}$, $\boldsymbol{H}_{c_{r_i}}^*$, Status, $\boldsymbol{v}^{\text{HIER}}, \boldsymbol{\omega}^{\text{HIER}}$ and optionally $\boldsymbol{v}^{\text{GLOBAL}}, \boldsymbol{\omega}^{\text{GLOBAL}}$

Table 4.2: Key message types that can be output by robot r_i using the SoNS control algorithm, along with the robot types that can receive them and their key contents. (Reprinted from (Zhu et al., 2024).)

Function name	Default equation
TargetCard (Eq. F1)	$\textsc{VertexCard}^*_{r_i} = \left\lvert r_j \in V(H^*_{r_i})\right\rvert$
Subgraph (Eq. F2)	$H^*_{c_{r_i}} = \left\{V(H^*_{c_{r_i}}),\ E(H^*_{c_{r_i}})\right\} = \left\{R^+_{H^*_{r_i}}(r^*_j) \cup r^*_j \in V(H^*_{r_i}),\ \ e_{ij} \lvert i \in V(H^*_{r_j})\right\}$ *Note: $V(G)$ and $E(G)$ denote the vertices and edges of graph G, respectively, and $R^+_G(v)$ denotes all vertices in the directed graph G reachable from vertex v.*
SumCard (Eq. F3)	$\textsc{VertexCard}_{r_i} = \sum_{c_{r_i} \in \boldsymbol{C}_{r_i}} \textsc{VertexCard}_{c_{r_i}} + 1$
SumHigh (Eq. F4)	$\textsc{VertexHigh}_{r_i} = \max(\textsc{VertexHigh}_{c_{r_i}} \lvert c_{r_i} \in \boldsymbol{C}_{r_i}) + 1$
Transform (Eq. F5)	$\boldsymbol{d}_{r_i a} = \boldsymbol{d}_{r_i r_j} + \mathrm{RT}(\boldsymbol{d}_{r_j a}, \boldsymbol{q}_{r_i r_j})$ $\boldsymbol{v}_{r_i} = \mathrm{RT}(\boldsymbol{v}_{r_j}, \boldsymbol{q}_{r_i r_j})$ $\boldsymbol{q}_{r_i a} = \mathrm{H}(\boldsymbol{q}_{r_i r_j}, \boldsymbol{q}_{r_j a})$ $\boldsymbol{\omega}_{r_i} = \mathrm{RT}(\boldsymbol{\omega}_{r_j}, \boldsymbol{q}_{r_i r_j})$ *Note: $RT(\boldsymbol{x}, \boldsymbol{y})$ is a function to rotate vector $\boldsymbol{y}$ by unit quaternion $\boldsymbol{x}$ using the Euler–Rodrigues formula, with the Euler parameters given by the coefficients of quaternions $\boldsymbol{y}^p = (0, \boldsymbol{y})$ and $\boldsymbol{x}$.* *$H(\boldsymbol{x}, \boldsymbol{y})$ takes the Hamilton product of two quaternions $\boldsymbol{x}$ and $\boldsymbol{y}$.*
Local (Eq. F6)	$\boldsymbol{v}^{\mathrm{LOCAL}}_{r_i} = \begin{cases} \boldsymbol{v}^{\mathrm{MAX}} \times \widehat{\boldsymbol{d}_{a\ r_i}} & \text{if } \delta < k_1 \\ \min\left(\boldsymbol{v}^{\mathrm{MAX}},\ -k_2 \log\left(\frac{\delta - k_1}{k_3 - k_1}\right)\right) \times \widehat{\boldsymbol{d}_{a\ r_i}} & \text{if } k_1 < \delta < k_3 \\ 0 & \text{otherwise} \end{cases}$
Hier (Eq. F7)	$\boldsymbol{v}^{\mathrm{HIER}}_{c_{r_i}} = \begin{cases} \boldsymbol{v}^{\mathrm{DEFAULT}} \times \widehat{\boldsymbol{d}^*_{c_{r_i} c_{r_i}'}} & \text{if } \left\lvert \boldsymbol{d}^*_{c_{r_i} c_{r_i}'} \right\rvert > k_4 \\ \boldsymbol{v}^{\mathrm{DEFAULT}} \times \frac{\left(\left\lvert \boldsymbol{d}^*_{c_{r_i} c_{r_i}'} \right\rvert - k_5\right)}{k_4 - k_5} \times \widehat{\boldsymbol{d}^*_{c_{r_i} c_{r_i}'}} & \text{if } k_5 < \left\lvert \boldsymbol{d}^*_{c_{r_i} c_{r_i}'} \right\rvert < k_4 \\ 0 & \text{otherwise} \end{cases}$
Global (Eq. F8)	$\boldsymbol{v}^{\mathrm{GLOBAL}}_{r_i} = \boldsymbol{v}^{\mathrm{MAX}} \times \widehat{\boldsymbol{d}_{a\ r_i}}$ *Note: δ denotes Euclidean distance from a to r_i and k_1-k_5 are scalar parameters to be tuned based on the hardware platform used.* *Equivalents of Eqs. 6-8 are used to calculate $\boldsymbol{\omega}^{LOCAL}_{r_i}, \boldsymbol{\omega}^{HIER}_{r_i}$, and $\boldsymbol{\omega}^{GLOBAL}_{r_i}$.* *If robot r_i is a brain, then $\boldsymbol{v}^{LOCAL}_{r_i}, \boldsymbol{v}^{HIER}_{r_i}, \boldsymbol{\omega}^{LOCAL}_{r_i}, \boldsymbol{\omega}^{HIER}_{r_i} = [0, 0, 0]$, and $\boldsymbol{v}^{GLOBAL}_{r_i}, \boldsymbol{\omega}^{GLOBAL}_{r_i}$ are output by the BrainProgram.*
Tar (Eq. F9)	$\boldsymbol{v}^*_{r_i} = \sum\ \boldsymbol{v}^{\mathrm{LOCAL}}_{r_i} + \boldsymbol{v}^{\mathrm{HIER}}_{r_i} + \boldsymbol{v}^{\mathrm{GLOBAL}}_{r_i} + \sum\ \boldsymbol{v}^{\mathrm{GLOBAL}}_{r_j}$ *with max. magnitude* $\lvert \boldsymbol{v}^{\mathrm{MAX}} \rvert$ $\boldsymbol{\omega}^*_{r_i} = \sum\ \boldsymbol{\omega}^{\mathrm{LOCAL}}_{r_i} + \boldsymbol{\omega}^{\mathrm{HIER}}_{r_i} + \boldsymbol{\omega}^{\mathrm{GLOBAL}}_{r_i} + \sum\ \boldsymbol{\omega}^{\mathrm{GLOBAL}}_{r_j}$ *with max. magnitude* $\lvert \boldsymbol{\omega}^{\mathrm{MAX}} \rvert$

Table 4.3: Default equations used in the SoNS control algorithm. (Reprinted from (Zhu et al., 2024).)

(Eq. F3) and SumHigh (Eq. F4), based on the estimated downstream vertex cardinalities and heights received from its children ($\text{VertexCard}^*_{c_{r_i}}$, $\text{VertexHigh}^*_{c_{r_i}}$), if it has any. It also updates its SoNS identifier and quality metric, either based on information received from its parent or, if it is currently a brain, according to its own information.

Vector transformation: Each robot r_i uses the function Transform (Eq. F5) to convert all input vectors received from any parent or child robot r_j into its own coordinate frame. This includes reference vectors $\boldsymbol{v}_{r_j}$, $\boldsymbol{\omega}_{r_j}$ and sensor information $\boldsymbol{d}_{r_j a}$, $\boldsymbol{q}_{r_j a}$ for any feature a, which are then used in other functions.

Update reference vectors for actuation instructions: Robot r_i uses the function Local (Eq. F6) to update its own local reference vectors, if it is not currently a brain; uses the function Hier (Eq. F7) to update hierarchical reference vectors to send to its children, if it has any; and, if it determines that a feature it sees requires a SoNS-wide reaction, uses the function Global (Eq. F8) to update a global reference to send to any robot connected to it (parent or child).

Update target velocities: If it is not currently a brain, robot r_i uses function Tar to update its own target velocities.

The following functions detailed in pseudocodes are related to feature classification and connection management in the SoNS control algorithm. The functions are used as follows:

Adding connections: Each robot r_i uses the Compete function (Alg. 1) to send, accept, or reject recruitment messages when a robot enters its field of view (FoV), determining whether that robot will become its parent, child, or neither. It compares its quality metric and other node attributes to those of the inquiring robot before deciding. Simultaneously, r_i uses the Bid function (Alg. 2) to adjust its quality metric for future iterations of Compete if necessary (see message types in Table 4.2).

Breaking connections: Each robot uses the Break function (Alg. 3) to remove connections with any robots that have left its field of view (FoV) or with which it has decided to break intentionally. If it disconnects from its parent, the robot updates its quality metric for future recruitment competitions and becomes a brain by default.

Algorithm 1 Compete function. (Reprinted from (Zhu et al., 2024).). If robot r_i receives a recruitment message from any robot f_{r_i} in its field of view, it will respond with acceptance or rejection. It rejects recruitment if it is currently a member of the SoNS of f_{r_i} or recently left it, or if its SoNS quality is higher than that of f_{r_i}; otherwise, it accepts. If robot r_i accepts, it updates its parent to f_{r_i}. Robot r_i also sends a recruitment message to any robot in its FoV that did not recently reject it. If robot r_i receives an acceptance from any robot f_{r_i}, it adds f_{r_i} to its set of children.

function COMPETE
$t^{\text{OLD}_{r_i}} \leftarrow t^{\text{OLD}_{r_i}} + 1$
for robot $f_{r_i} \in \boldsymbol{F}_{r_i}$ **do**
  if ACCEPTRECRUITMENT message is received **then**
    $\boldsymbol{C}_{r_i} \leftarrow \boldsymbol{C}_{r_i} + f_{r_i}$
  else if RECRUIT message is received **then**
    if $\text{SONSID}_{f_{r_i}} = \text{SONSID}_{r_i}$ **then**
      send REJECTRECRUITMENT message
    else if $(\text{SONSID}_{f_{r_i}} = \text{SONSID}_{r_i}^{\text{OLD}}) \wedge (t_{r_i}^{\text{OLD}} \leq \text{VERTEXHIGH}_{r_i})$ **then**
      send REJECTRECRUITMENT message
    else if $\text{QUALITY}_{r_i} \geq \text{QUALITY}_{f_{r_i}}$ **then**
      send REJECTRECRUITMENT message
    else
      send ACCEPTRECRUITMENT message
      $p_{r_i} \leftarrow f_{r_i}$
    end if
  end if
  if no REJECTRECRUITMENT message was received in the last $t^{\text{REJECTTIMER}}$ time steps **then**
    send RECRUIT message
  end if
end for
end function

Algorithm 2 Bid function. (Reprinted from (Zhu et al., 2024).) If robot r_i updates its bid for recruitment competitions, it breaks its link with its parent p_{r_i} and updates its QUALITY according to its new bid value.

function BID
if SPLIT message is received from p_{r_i} **then**
  BIDVALUE $\leftarrow$ randomly generated
  $p_{r_i} \leftarrow$ empty
  $\text{QUALITY}_{r_i} \leftarrow$ BIDVALUE
end if
end function

This function also updates the robot's node attributes accordingly if its connection with its parent is lost, either intentionally or accidentally.

Algorithm 3 Break function. (Reprinted from (Zhu et al., 2024).) If robot r_i no longer has its parent or one of its children in its FoV, it removes its connection to that robot. If the parent of robot r_i has been updated or made empty by any function, robot r_i updates its SonsID and Quality.

1: **function** Break
2: **for** $c_{r_i} \in \boldsymbol{C}_{r_i}$ **do**
3: **if** c_{r_i} has not been in the field of view of r_i in the last $t^{\text{BreakTimer}}$ time steps **then**
4: $\boldsymbol{C}_{r_i} \leftarrow \boldsymbol{C}_{r_i} - c_{r_i}$
5: **end if**
6: **end for**
7: **if** p_{r_i} has not been in the field of view of r_i in the last $t^{\text{BreakTimer}}$ time steps **then**
8: $p_{r_i} \leftarrow$ empty
9: **end if**
10: **if** p_{r_i} has been modified during this function or functions Compete 1 or Bid 2 **then**
11: **if** p_{r_i} is not empty **then**
12: $\text{SonsID}_{r_i} \leftarrow \text{SonsID}_{p_{r_i}}$
13: $\text{Quality}_{r_i} \leftarrow \text{Quality}_{p_{r_i}}$
14: **else**
15: $\text{SonsID}_{r_i} \leftarrow \text{RobotID}_{r_i}$
16: **if** p_{r_i} was made empty during this function or function Compete **then**
17: $\text{Quality}_{r_i} \leftarrow$ randomly generated
18: **end if**
19: **end if**
20: $\text{SonsID}_{r_i}^{\text{OLD}} \leftarrow$ former SonsID_{r_i}
21: $t_{r_i}^{\text{OLD}} \leftarrow 0$
22: **end if**
23: **end function**

Managing connections: Each robot r_i manages and (re)allocates its children at each step by using the Allocate function (Alg. 4). This involves assigning children to nodes within its own target graph or reallocating them to be managed by its parent, as necessary. These (re)allocation actions are guided by comparing the current states of r_i and its children with the target states specified in the target graph (see HandOver message in Table 4.2).

Feature classification: Each robot that is not currently a brain uses the Classify function (Alg. 5) to categorize environmental features as meriting a local reaction, a SoNS-wide (global) reaction, or forwarding to its parent for handling further

Algorithm 4 Allocate function. (Reprinted from (Zhu et al., 2024).) Robot r_i allocates each of its current children to one of its target child positions $c^*_{r_i} \in \boldsymbol{C}^*_{r_i}$ or to be handed over to its parent.

function ALLOCATE
TARGETMATCH($\boldsymbol{C}_{r_i} \cup r_i, C^*_{p_{r_i}}$) {Self and children as sources; targets of parent and self}
for source robot in $\boldsymbol{\mathcal{CB}'}$ returned by TARGETMATCH **do**
  if source robot matches the same target node in $\boldsymbol{\mathcal{CB}'}$ that is matched by r_i **then**
    add source robot to the set $\boldsymbol{C}'_{r_i}$ to be used in the next allocation steps
  else
    assign source robot to parent and send HANDOVER message
  end if
end for
for source robot r_j in $\boldsymbol{C}'_{r_i}$ **do**
  if SHORTESTDISTANCE($p_{r_i}, p^*_{r_i}, r_j$) < SHORTESTDISTANCE($p_{r_i}, p^*_{r_i}, r_i$) **then**
    assign source robot to parent and send HANDOVER message
  end if
end for
TARGETMATCH($\boldsymbol{C}'_{r_i}, \boldsymbol{C}^*_{r_i}$) {Any remaining children as sources; targets of self}
for source robot in $\boldsymbol{\mathcal{CB}'}$ **do**
  if source is uniquely matched to target in $\boldsymbol{\mathcal{CB}'}$ **then**
    assign source robot to target node
  else if multiple source robots r_j are matched to the same target r^*_j **then**
    select the source robot r_j with the lowest SHORTESTDISTANCE(r_i, r^*_j, r_j)
    assign remaining source robots to the selected r_j and send HANDOVER messages
  else if one source is matched to multiple targets **then**
    send the list of targets to the source robot for use in its ALLOCATE function
  else
    assign source robot to parent and send HANDOVER message
  end if
end for{Algorithm continues on the next page}

Algorithm 4 Allocate function *(cont'd).* (Reprinted from (Zhu et al., 2024).)

for $c_{r_i} \in \boldsymbol{C}_{r_i}$ **do**
  if the connection from r_i to c_{r_i} spatially intersects with another detected connection **then**
    if the intersecting connection is with another c_{r_i} **then**
      swap their node assignments
    else
      send HANDOVER message towards the parent of the detected connection
    end if
  end if
end for
end function
function SHORTESTDISTANCE(r_k, r_l, r_m)
    return $\boldsymbol{d}_{r_k r_l} \cdot \frac{\boldsymbol{d}^*_{r_k r_m}}{\|\boldsymbol{d}^*_{r_k r_m}\|}$
end function
function TARGETMATCH $\boldsymbol{C}_{r_i}$,$\boldsymbol{C}^*_{r_i}$
define $\boldsymbol{\mathcal{CB}}$ as all unique combinations of entries $c_{r_i} \in \boldsymbol{C}_{r_i}$ and $c^*_{r_i} \in \boldsymbol{C}^*_{r_i}$
$\boldsymbol{\mathcal{S}}^{\mathrm{D}} \leftarrow \boldsymbol{d}_{r_i c_{r_i}}\ \forall c_{r_i} \in \boldsymbol{C}_{r_i}$ {Source displacements}
$\boldsymbol{\mathcal{S}}^{\mathrm{CARD}} \leftarrow \mathrm{VERTEXCARD}_{c_{r_i}} \forall c_{r_i} \in \boldsymbol{C}_{r_i}$ {Source downstream cardinalities}
$\boldsymbol{\mathcal{T}}^{\mathrm{D}} \leftarrow \boldsymbol{d}^*_{r_i c_{r_i}} \forall c^*_{r_i} \in \boldsymbol{C}^*_{r_i}$ {Target displacements}
$\boldsymbol{\mathcal{T}}^{\mathrm{CARD}} \leftarrow \mathrm{VERTEXCARD}^*_{c_{r_i}}\ \forall c^*_{r_i} \in \boldsymbol{C}^*_{r_i}$ {Target downstream cardinalities}
$\boldsymbol{\mathcal{W}}^{\mathrm{D}} \leftarrow ||\mathcal{S}^{\mathrm{D}}_j - \mathcal{T}^{\mathrm{D}}_j||^2\ \forall\ (c_{r_i}, c^*_{r_i}) \in \boldsymbol{\mathcal{CB}}$ {Displacement costs}
$\mathcal{S}^{\mathrm{CAPAC}}, \mathcal{T}^{\mathrm{CAPAC}} \leftarrow ||\mathcal{S}^{\mathrm{CARD}}_j||, ||\mathcal{T}^{\mathrm{CARD}}_j||\ \forall\ (c_{r_i}, c^*_{r_i}) \in \boldsymbol{\mathcal{CB}}$ {Source capacity and Target capacity}
construct a network flow graph G_{FLOW} using the defined matrices
apply the NETWORKFLOW algorithm (Cormen et al., 2001; Wayne, 1999) to G_{FLOW} to obtain the maximum flow rate network G'_{FLOW}
use G'_{FLOW} to select the source in $\boldsymbol{\mathcal{S}}^{\mathrm{D}}, \boldsymbol{\mathcal{S}}^{\mathrm{CARD}}$ to match to each target in $\boldsymbol{\mathcal{T}}^{\mathrm{D}}, \boldsymbol{\mathcal{T}}^{\mathrm{CARD}}$

return matches $(c_{r_i}, c^*_{r_i}) \in \boldsymbol{\mathcal{CB}'}$ {Matches are candidates for node assignments}

end function

upstream in the hierarchy. If the robot is currently a brain, it processes features according to its BrainProgram.

Algorithm 5 Classify function. (Reprinted from (Zhu et al., 2024).) Robot r_i classifies features it detects directly or receives from any children into features that: can be handled locally, can be handled by the parent, or require a SoNS-wide reaction.

```
function Classify
for a ∈ A_{r_i} do
    if a is a feature type that can be avoided locally e.g., a^SmallObstacle then
        use a in function Local Eq. F6
    else if a is a feature type that requires a SoNS-wide reaction then
        use a in function Global Eq. F8
        else forward a to p_{r_i}
    end if
end for
end function
```

4.2.3 Mission-specific modules of the SoNS control algorithm

Most functions within the SoNS algorithm have default versions but can be adjusted as parameters to meet specific task requirements. However, two key functions, BrainProgram and Bid, are designed to be mission-specific, enabling the entire SoNS to operate as if it were a single robot with a reconfigurable structure. The versions of these functions used in this study are provided in the pseudocode below. Table 4.4 summarizes the primary functions and variables utilized within the BrainProgram.

4.3 Convergence and stability analysis

To verify our SoNS architecture, we consider a distance-based positioning approach (in which every robot can reference a single other nearby robot) and a reactive control law to establish and maintain the target relative positions using onboard measurements. This simple approach can be understood as a performance baseline for the tracking of target positions within a SoNS.

Function name	Default equation
StopMoving	$\boldsymbol{v}^*_{r_i} = (0,0,0)$, $\boldsymbol{\omega}^*_{r_i} = (0,0,0)$
MoveForward	$\boldsymbol{v}^*_{r_i} = (0.2,0,0)$, $\boldsymbol{\omega}^*_{r_i} = (0,0,0)$
MoveToObject	$\boldsymbol{v}^*_{r_i} = (a_x, a_y, 0)$, $\boldsymbol{\omega}^*_{r_i} = (0,0,a_r)$
	Note: a_x, a_y, a_r are calculated for movement towards and alignment with object $\boldsymbol{a}$.

Variable	Description
$\boldsymbol{H}^{*\text{Label}}$	labeled target graph
$\boldsymbol{\mathcal{H}}^{\text{Label}}$	labeled matrix of target graphs $\boldsymbol{H}^*$ with attributes

Table 4.4: Key functions and variables used in the BrainProgram modules. (Reprinted from (Zhu et al., 2024).)

Algorithm 6 Brain Program. (Reprinted from (Zhu et al., 2024).) Primary module through which a multi-robot SoNS can effectively be programmed as if it were a single robot with a reconfigurable morphology.

Input: $\boldsymbol{A}_{r_i}$, Status, optionally VertexCard$_{r_i}$ and VertexCard$^*_{r_i}$
Output: $\boldsymbol{v}^*_{r_i}, \boldsymbol{\omega}^*_{r_i}, \boldsymbol{H}^*_{r_i}$, optionally Status or other outputs for $c_{r_i} \in \boldsymbol{C}_{r_i}$

Alg. 6a Variation used in the **Establishing self-organizing hierarchy** mission. When robot r_i is the brain, it tries to stay in its current position and does not change its target graph.

function BrainProgram
$\boldsymbol{H}^*_{r_i} \leftarrow \boldsymbol{H}^{*\text{Default}}$
StopMoving
end function

Alg. 6b Variation used in the **Balancing global and local goals** mission. When robot r_i is the brain, it tries to move forward until it detects the final destination object and does not change its target graph.

function BrainProgram
$\boldsymbol{H}^*_{r_i} \leftarrow \boldsymbol{H}^{*\text{Default}}$
if $\boldsymbol{a}^{\text{Destination}} \in \boldsymbol{A}_{r_i}$ **or** Status = Destination **then**
  Status ← Destination
  if $\boldsymbol{a}^{\text{Destination}}$ is near the target relative position **then** StopMoving
  else MoveToObject
  end if
  else MoveForward
end if
end function

Alg. 6c Variation used in the **Collective sensing and actuation** mission. When robot r_i is the brain, it tries to move forward until it detects the final destination object. If it detects any walls, it selects the target graph with the widest shape that will still fit between the walls.

function BrainProgram
$\boldsymbol{A}^{\text{LeftWall}} \leftarrow$ the set of all $\boldsymbol{a}^{\text{Wall}} \in \boldsymbol{A}_{r_i}$ that are to the left of the body frame of r_i
$\boldsymbol{A}^{\text{RightWall}} \leftarrow$ the set of all $\boldsymbol{a}^{\text{Wall}} \in \boldsymbol{A}_{r_i}$ that are to the right of the body frame of r_i
if $\boldsymbol{a}^{\text{Destination}} \in \boldsymbol{A}_{r_i}$ **or** Status = Destination **then**
  Status ← Destination
  $\boldsymbol{H}^*_{r_i} \leftarrow \boldsymbol{H}^{*\text{Final}}$
  if $\boldsymbol{a}^{\text{Destination}}$ is near the target relative position **then** StopMoving
  else MoveToObject
  end if
else if $(\boldsymbol{A}^{\text{LeftWall}} \neq \emptyset) \wedge (\boldsymbol{A}^{\text{RightWall}} \neq \emptyset)$ **or** Status = FitsBetween **then**
  Status ← FitsBetween
  estimate ShortestDistance from any $\boldsymbol{a}^{\text{Wall}} \in \boldsymbol{A}^{\text{LeftWall}}$ to any $\boldsymbol{a}^{\text{Wall}} \in \boldsymbol{A}^{\text{RightWall}}$
  $\boldsymbol{\mathcal{H}}^{\text{FitsBetween}} \leftarrow$ the set of all $\boldsymbol{H}^* \in \boldsymbol{\mathcal{H}}^{\text{DifferentWidths}}$ with Width < ShortestDistance
  $\boldsymbol{H}^*_{r_i} \leftarrow \boldsymbol{H}^* \in \boldsymbol{\mathcal{H}}^{\text{FitsBetween}}$ with maximum Width
  MoveForward
else
  $\boldsymbol{H}^*_{r_i} \leftarrow \boldsymbol{H}^{*\text{Default}}$
  MoveForward
end if
end function {Algorithm continues on the next page}

Algorithm 6 Brain Program *(cont'd).* (Reprinted from (Zhu et al., 2024).)

Alg. 6d Variation used in the **Binary decision-making** mission. When robot r_i is the brain, it tries to move forward until it detects the final destination object. If it detects a wall in front of it, it stops moving unless it detects an opening in the wall. If it detects an opening, it switches to a target graph with a narrow shape and tries to move forward.

function BrainProgram.
if $\boldsymbol{a}^{\text{Destination}} \in \boldsymbol{A}_{r_i}$ **or** Status = Destination **then**
  Status ← Destination
  $\boldsymbol{H}^*_{r_i} \leftarrow \boldsymbol{H}^{*\text{Final}}$
  if $\boldsymbol{a}^{\text{Destination}}$ is near the target relative position **then** StopMoving
  else MoveToObject
  end if
else if $\boldsymbol{A}_{r_i}$ includes a $\boldsymbol{a}^{\text{LeftEdge}}$ that is positioned to the left of a $\boldsymbol{a}^{\text{RightEdge}}$ **or** Status = Narrow **then**
  Status ← Narrow
  $\boldsymbol{H}^*_{r_i} \leftarrow \boldsymbol{H}^{*\text{Narrow}}$
  MoveForward
else
  $\boldsymbol{H}^*_{r_i} \leftarrow \boldsymbol{H}^{*\text{Default}}$
  MoveForward
end if
end function

Alg. 6e Variation used in the **Splitting and merging** mission. When robot r_i is a brain with default status and its target graph is incomplete, it instructs a robot in its SoNS to temporarily split. When robot r_i is a brain with Search status, it follows walls in the environment until its target graph is complete, then retraces its previous path.

function BrainProgram
if Status=Search **then**
  $\boldsymbol{H}^*_{r_i} \leftarrow \boldsymbol{H}^{*\text{Search}}$
  if $\text{VertexCard}_{r_i} < \text{VertexCard}^*_{r_i}$ **then**
    if $\boldsymbol{a}^{\text{Wall}} \in \boldsymbol{A}_{r_i}$ **then**
      FollowWall
      else MoveForward
    end if
    else RetracePreviousPath
  end if
else
  $\boldsymbol{H}^*_{r_i} \leftarrow \boldsymbol{H}^{*\text{Default}}$
  else StopMoving
  if Status=Default **and** $\text{VertexCard}_{r_i} < \text{VertexCard}^*_{r_i}$ after t^{Check} time steps **then**
    construct Split message with added instruction to set Status = Search at destination
    send Split message to child c_{r_i} that is upstream from target destination r^*_j
    Status ← Waiting
  else if $\text{VertexCard}_{r_i} \geq \text{VertexCard}^*_{r_i}$ **then**
    Status ← Default
  end if
end if
end function

Algorithm 7 Bid function variations. (Reprinted from (Zhu et al., 2024).). The default BID function (Alg. 2) is used in most experiment setups in this study. The exception is the following two mission types, in which it is used as a mission-specific parameter.

Alg. 7a Variation used in the **Binary decision-making** mission. Robot r_i increases its QUALITY if the features in its environment are favorable for the next mission step: specifically, if it detects that it is near an opening in a wall blocking its path, it increases its QUALITY according to the detected width of the wall opening.

function BID
if $\boldsymbol{A}$ includes a $\boldsymbol{a}^{\text{LEFTEDGE}}$ that is positioned to the left of a $\boldsymbol{a}^{\text{RIGHTEDGE}}$ **then**
estimate DISTANCE from $\boldsymbol{a}^{\text{LEFTEDGE}}$ to $\boldsymbol{a}^{\text{RIGHTEDGE}}$
BIDVALUE$\leftarrow$DISTANCE
$p_{r_i} \leftarrow$ empty
$\text{QUALITY}_{r_i} \leftarrow \text{QUALITY}_{r_i} + \text{BIDVALUE}$
end if
end function

Alg. 7b Variation used in the **Splitting and merging** mission. Robot r_i decreases its QUALITY slightly when it is instructed to split. The effect is that, when robot r_i later returns and the two SoNS re-merge, robot r_i will lose the recruitment competition and will revert to being a child in the SoNS from which it split.

function BID
if SPLIT message is received from p_{r_i} **then**
BIDVALUE$\leftarrow -0.01$
$p_{r_i} \leftarrow$ empty
$\text{QUALITY}_{r_i} \leftarrow \text{QUALITY}_{r_i} + \text{BIDVALUE}$
end if
end function

To provide theoretical analyses and mathematical proofs regarding the convergence of the position errors and closed-loop stability of position tracking in a SoNS with respect to the control inputs, we represent a SoNS as a system of robot pairs, use a proportional control law that is strictly reactive, and derive the leader–follower tracking kinematics assuming the robots cannot access any global information or external reference frame, instead using only local sensing of relative information that is available in our real setup.

Specifically, we represent a SoNS of n robots as a multi-robot formation and decompose it into $n-1$ sub-formations of leader-follower pairs. To study the interactions and stability bounds of a SoNS, we analyze the local leader-follower formation tracking problems of these pairs. We use a standard reactive control law (Krick et al., 2009; Oh and Ahn, 2011) to generate appropriate inputs for the follower robot, in order to maintain the desired relative position with respect to its leader. This strictly reactive control law is the same approach we use in our real robot experiments; it establishes a performance baseline for the tracking of target positions within a SoNS. We assess the convergence and closed-loop stability of position tracking in the system, utilizing the *leader-to-formation stability* notion (Tanner et al., 2004).

At the end of this section, we discuss other types of control laws that could be combined with the SoNS architecture to improve performance beyond the baseline established using a strictly reactive control law.

4.3.1 Modeling

Consider a SoNS with n robots (including both aerial robots and ground robots). The translational motion in $\mathbb{R}^2$ space of a robot R_i, $i \in \{1, 2, \ldots, n\}$, is governed by(Oh and Ahn, 2011):

$$\dot{\vec{p}}_i = \vec{u}_i \tag{4.1}$$

where $\vec{p}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}^T \in \mathbb{R}^2$ is the absolute position in a global coordinate system $\mathcal{F}_\mathcal{I}$ used for analysis and $\vec{u}_i \in \mathbb{R}^2$ is the control input.

4.3.2 Control of a leader-follower pair

We first analyze the interactions of a single leader-follower pair, before extending to multiple leader-follower pairs.

Leader-follower kinematics

We derive the leader-follower kinematics for a formation consisting of a single robot pair. Consider the leader-follower setup in Fig. 4.4, where $\vec{p}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}^T$ is the position of the leader robot R_i, $\vec{p}_j = \begin{bmatrix} x_j & y_j \end{bmatrix}^T$ is the position of the follower robot R_j, $\vec{d}_{ij} \in \mathbb{R}^2$ is the desired displacement of R_j w.r.t. R_i and is defined according to the control outputs of the SoNS software, all expressed in the global coordinate frame used for the analysis. From this setup, we can define $\vec{z}_{ij} = \vec{p}_i - \vec{p}_j$ as the displacement of the follower R_j with respect to the leader R_i, $\|\vec{z}_{ij}\| \in \mathbb{R}$ as the Euclidean norm of that displacement, $\|\vec{d}_{ij}\| = \|\vec{d}_{ji}\| \in \mathbb{R}$ as the Euclidean norm of the desired displacement $\vec{d}_{ij}$, $\vec{p}_j{}^d = \vec{p}_i - \vec{d}_{ij}$ as the desired position of R_j, and $\vec{e}_{ij} = \vec{z}_{ij} - \vec{d}_{ij} \in \mathbb{R}^2$ as the formation tracking error.

With this notation, the kinematics of a leader-follower pair can be defined as

$$\begin{aligned} \dot{\vec{e}}_{ij} &= \dot{\vec{p}}_i - \dot{\vec{p}}_j \\ &= \vec{u}_i - \vec{u}_j \end{aligned} \tag{4.2}$$

Remark 3: Note that robots in a SoNS are assumed to not have access to any global position information or external reference frame. Each robot can only access the relative positions $\vec{z}_{ij}$ of its neighbors $j \in \mathcal{N}_i$, with respect to its own local reference frame. Global position information is used exclusively for analysis.

Problem formulation

The control law that generates inputs for the follower robot to move from its current position $\vec{p}_j$ to the desired position $\vec{p}_j{}^d$ can be expressed as follows.

Problem definition: The goal is to ensure that the formation tracking of the

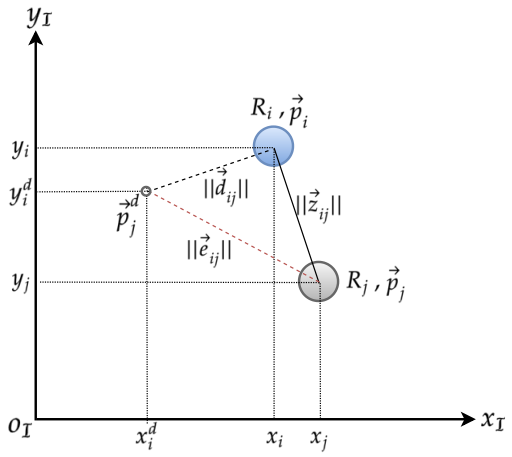


Figure 4.4: An example leader-follower pair. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

follower robot R_j adheres to the motion model given in Eq. (4.1) and conforms to the leader-follower kinematics given in Eq. (4.2), using the following control law $\vec{u}_j$ for the follower robot:

$$\vec{u}_j = f(t, \vec{z}_{ij}, \vec{d}_{ij}) \in \mathbb{R}^2 \tag{4.3}$$

where $\vec{z}_{ij}$ is the displacement with respect to the leader and $\vec{d}_{ij}$ is the desired displacement. The function $f(\cdot)$ generates the required control inputs to move the follower robot from its current position $\vec{p}_j$ to the desired position $\vec{p}_j{}^d$, such that the norm of the formation tracking error $\|\vec{e}_{ij}\| \to 0$ and $\|\vec{z}_{ij}\| \to \|\vec{d}_{ij}\|$.

Control law design and analysis

Based on the leader-follower kinematics given in Eq. (4.2), we use the following standard reactive controller (Krick et al., 2009; Oh and Ahn, 2011) for the follower robot R_j:

$$\begin{aligned} \vec{u}_j &= \boldsymbol{K}^j(\vec{z}_{ij} - \vec{d}_{ij}) \\ &= \boldsymbol{K}^j \vec{e}_{ij} \end{aligned} \tag{4.4}$$

where $\boldsymbol{K}^j = (\boldsymbol{K}^j)^T = \begin{bmatrix} k_1^j & 0 \\ 0 & k_2^j \end{bmatrix}$ is a symmetric positive-definite matrix, the constants k_1^j and k_2^j are control gains, and $\vec{u}_j \in \mathbb{R}^2$ is the input for maintaining the desired displacement $\|\vec{d}_{ij}\|$ of the follower with respect to the leader. The control law is independent of any global position information; it uses only relative position information between the follower and the leader, making it independent of any global position information.

Input-to-state stability and bounding of formation tracking errors

We first analyze the stability properties of the leader-follower kinematics Eq. (4.2) in the case of zero external input based on the following definition of Lyapunov stability.

Definition 1: (*exponential stability* (Khalil and Grizzle, 2002)) For a time-invariant system $\dot{x} = f(x)$ where $f : \mathcal{D} \subset \mathbb{R}^n \to \mathbb{R}^n$ is a locally Lipschitz function and $x = 0 \in \mathcal{D}$ is an equilibrium point of the system, exponential stability can be established through the use of a Lyapunov function $V(x)$ that satisfies the following conditions:

- $V(x)$ is $\mathcal{C}^1$ (i.e., continuously differentiable)
- $c_1\|x\|^a \leq V(x) \leq c_2\|x\|^a$
- and $\dot{V}(x) \leq -c_3\|x\|^a$

where $a > 0$, $c_1 > 0$, $c_2 > 0$, and $c_3 > 0$.

Theorem 1: For a leader-follower robot pair with the motion model Eq. (4.1), an initial condition $\vec{p}_i(0) \neq \vec{p}_j(0)$, and control input $\vec{u}_i = 0$, the controller Eq. (4.4) guarantees exponential stability of $\vec{e}_{ij} \to 0$ as $t \to \infty$, in the system given in Eq. (4.2).

Proof 1: By substituting the control law Eq. (4.4) into the motion model Eq. (4.1), we obtain the following closed-loop system:

$$\dot{\vec{p}}_j = \boldsymbol{K}^j \vec{e}_{ij} \tag{4.5}$$

The leader-follower kinematics Eq. (4.2) can then be expressed as

$$\dot{\vec{e}}_{ij} = -\boldsymbol{K}^j \vec{e}_{ij} + \vec{u}_i \tag{4.6}$$

We consider the Lyapunov function candidate $V_1 = \frac{1}{2}\vec{e}_{ij}^T \vec{e}_{ij}$ that satisfies the following condition:

$$c_1^j \|\vec{e}_{ij}\| \leq V_1 \leq c_2^j \|\vec{e}_{ij}\| \tag{4.7}$$

where $a_j = 2$, $0 < c_1^j = \min(k_1^j, k_2^j)$, and $0 < c_2^j = \max(k_1^j, k_2^j)$. Note that the constants k_1^j and k_2^j are the gains of the matrix $\boldsymbol{K}^j$ defined in Eq. (4.4). Then, differentiating Eq. (4.7) with respect to time yields

$$\begin{aligned} \dot{V}_1 = &\quad \vec{e}_{ij}^T \dot{\vec{e}}_{ij} \\ = &\quad \vec{e}_{ij}^T (\vec{u}_i - \boldsymbol{K}^j \vec{e}_{ij}) \\ = &-\vec{e}_{ij}^T \boldsymbol{K}^j \vec{e}_{ij} + \vec{e}_{ij}^T \vec{u}_i \\ \leq &-2c_1^j \|\vec{e}_{ij}\|^2 + \|\vec{e}_{ij}\| \|\vec{u}_i\| \leq -c_3^j \|\vec{e}_{ij}\|^2 \leq 0, \ \forall \ \|\vec{e}_{ij}\| \geq \frac{\|\vec{u}_i\|}{2c_1^j \theta} \end{aligned} \tag{4.8}$$

where $c_3^j \triangleq 2c_1^j(1-\theta)$, $\theta \in (0,1)$.The proof is concluded by substituting $\vec{u}_i$.

With minor modification and following the lines of (Seibert and Suarez, 1990), the same proof can be used also to show that if the control input $\vec{u}_i \to 0$ then the formation tracking error $\|\vec{e}_{ij}\|$ converges to 0.

If the leader is moving with other velocity regimes, then there is a lower bound that the formation tracking error can attain, according to $\vec{u}_i$. In order to ensure that the quadrotors maintain stability within some flight safety requirements, it is crucial to bound the error amplitudes in the worst-case scenario. We use the input-to-state stability (ISS) notion (Isidori, 1985; Khalil and Grizzle, 2002; Sontag et al., 1995) to establish an upper limit for the formation tracking error and an upper limit for the admissible leader input $\vec{u}_i$ that can maintain flight safety at all times. We then use the analysis of the formation's ISS to establish a link between the magnitude of the leader's input and the evolution of the formation tracking errors, i.e., the error dynamics given in Eq. (4.6).

Definition 2: (*input-to-state–stability* (Sontag et al., 1995)) Let a leader-follower pair be input-to-state stable. Then, there is a class $\mathcal{KL}$ function β and a class $\mathcal{K}$ function γ such that, for any initial formation tracking error $\vec{e}_{ij}(0)$ and for any bounded input of the leader $\vec{u}_i(t)$, the solution $\vec{e}_{ij}(t)$ exists for all $0 \leq t$ and satisfies the following inequality (Tanner et al., 2004):

$$\|\vec{e}_{ij}(t)\| \leq \beta_{ij}(q,t) + \gamma_{ij}(r) \tag{4.9}$$

where the functions $\beta_{ij}(q,t)$ and $\gamma_{ij}(r)$ are transient and asymptotic ISS gain functions, respectively. These functions help to measure the affect of the initial conditions and the leader's input on the formation tracking errors (Tanner et al., 2002).

In order to further analyze the ISS properties of the leader-follower pair, we treat the error dynamics given by Eq. (4.6) as a perturbed system and derive an upper bound on the error norm $\|\vec{e}_{ij}(t)\|$. This bound provides a measure of the rate at which the formation tracking error converges and indicates that it is ISS with respect to the leader's velocity $\vec{u}_i$ (Isidori, 1985; Khalil and Grizzle, 2002; Tanner et al., 2002). Using the initial error norm $\|\vec{e}_{ij}(0)\|$ we can then rewrite the inequality Eq. (4.9) as

$$\begin{aligned} \|\vec{e}_{ij}(t)\| &\leq \beta_{ij}(\|\vec{e}_{ij}(0)\|, t) + \gamma_{ij}\left(\sup_{0\leq\tau\leq t}\|\vec{u}_i(\tau)\|\right) \\ &\leq \hat{\beta}_{ij}\|\vec{e}_{ij}(0)\| e^{-\frac{c_3^j}{c_2^j a_j}t} + \hat{\gamma}_{ij}\sup_{\tau\leq t}\|\vec{u}_i(\tau)\| \end{aligned}, \tag{4.10}$$

where the terms $\hat{\beta}_{ij}$ and $\hat{\gamma}_{ij}$ are gain estimates that provide insight into the relationship between the initial error, the leader's input, and the observed interconnection errors observed. They are defined as

$$\hat{\beta}_{ij} \triangleq \left(\frac{c_2^j}{c_1^j}\right)^{\frac{1}{a_j}}, \qquad \hat{\gamma}_{ij} \triangleq \frac{c_2^j}{c_1^j \theta} \tag{4.11}$$

By substituting the gain estimates from Eq. (4.11) into Eq. (4.10), we obtain

$$\|\vec{e}_{ij}(t)\| \leq \left(\frac{c_2^j}{c_1^j}\right)^{\frac{1}{a_j}} \|\vec{e}_{ij}(0)\| e^{-\frac{c_3^j}{c_2^j a_j} t} + \frac{c_2^j}{c_1^j \theta} \sup_{\tau \leq t} \|\vec{u}_i(\tau)\| \tag{4.12}$$

We use the formation ISS measure as a metric to help provide an upper bound on the leader's input and ensure that the formation remains within desired specifications. Additionally, we use it to compare the stability properties of different formation shapes and connection schemes.

Definition 3: (*formation ISS measure* (Tanner et al., 2002)) Consider a leader-follower pair that is ISS with gain functions $\beta_{ij}(q,t)$ and $\gamma_{ij}(r)$. Assume that $\gamma_{ij}(r) \in \mathcal{C}^1$ (i.e., it is continuously differentiable) and let $\mathcal{U} \subseteq \mathbb{R}^n$ be a compact neighborhood of the origin containing all $\vec{u}_i \in \mathcal{U}$ that are of interest. The formation tracking error $\vec{e}_{ij}$ always satisfies the following inequality if the leader-follower pair is ISS:

$$\lim_{t \to \infty} \|\vec{e}_{ij}(t)\| \leq \gamma_{ij}(r) \tag{4.13}$$

If we consider a specification such as the first leader's input bounded inside a unit sphere where $r = 1$, we can derive the formation's ISS performance measure $P_{ISS} \in [0,\ 1]$, based on the performance measure of leader-to-follower stability P_{LFS} in (Tanner et al., 2004), as follows:

$$P_{ISS} \triangleq \frac{1}{\gamma_{ij}(1)} \tag{4.14}$$

Formation tracking error with bounded leader inputs

Consider a leader-follower pair as depicted in Fig. 4.4, where the initial positions are $\vec{p}_i(0) = [5,\ 10]^T$ and $\vec{p}_j(0) = [0,\ 0]^T$, the desired displacement vector is given as $\vec{d}_{ij} = [3,\ 4]^T$, and the controller gain matrix $\boldsymbol{K}^j$ has $k_1^j = 5$ and $k_2^j = 5$.

Remark 4: In the real-robot and simulated experiments in this study, we use a strictly reactive control law; there is no feedforward control nor preview of the reference signal. We use this reactive control with the aim of establishing a perfor-

mance baseline for the tracking of target positions within a SoNS—in other words, to study the lower bounds of performance that the formation tracking error can attain. For a discussion of incorporating feedforward information in SoNS for improved performance, see the end of this section.

Under a stationary leader (i.e., $\vec{u}_i = [0, 0]$), the formation tracking error exponentially converges to zero (see Fig. 4.5a). The upper bound defined in Eq. (4.10) decays with respect to the initial formation tracking error and the lower bound defined in Eq. (4.8) is always 0 (the error norm converges to 0).

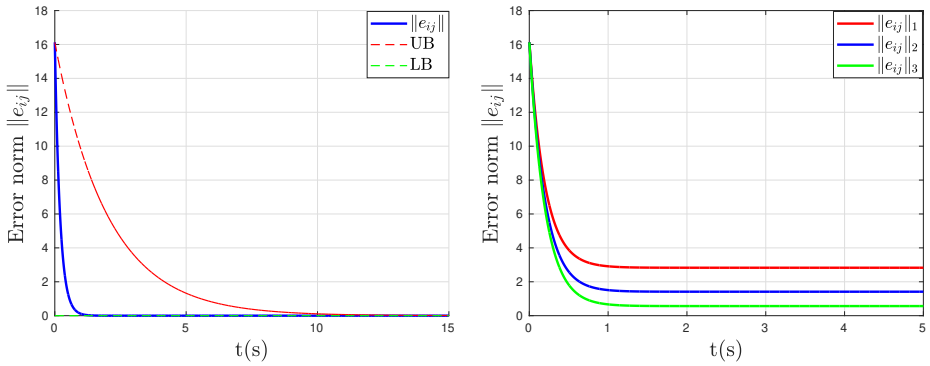


Figure 4.5: Norm of the formation tracking error without feed-forward information. (a) Error norm under a stationary leader, with the upper bound (UB) and lower bound (LB). Note that the lower bound line (green) is along the bottom of the graph. (b) Error norm under a moving leader with different velocity regimes. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

For cases of moving leaders, we consider the lower bound that the error can attain under velocity regimes that span the conditions occurring in our real robot experiments. We consider the example error norms $\|e_{ij}\|_1$, $\|e_{ij}\|_2$, and $\|e_{ij}\|_3$, which represent the formation tracking errors under leader velocities $\vec{u}_i^1 = [10,\ 10]^T$, $\vec{u}_i^2 = [5,\ 5]^T$, and $\vec{u}_i^3 = [2,\ 2]^T$, respectively. Under these velocity regimes, the error norms converge to a lower bound (see Fig. 4.5b). For instance, when the leader velocity is $\vec{u}_i^2$ and $\theta = 0.5$, the norm of the formation tracking error $\|e_{ij}\|_2$ converges to a lower bound around 1.5.

4.3.3 Control of multiple leader-follower pairs

We also assess how stability bounds are propagated in formations composed of multiple leader-follower pairs.

Graph theory preliminaries

For a SoNS, we define a target topology $G = (V, E, D)$, where $V = \{v_1, \ldots, v_n\}$ is a set of vertices, $E = \{\xi_{ij} = (\overrightarrow{v_j, v_i}) \mid v_j, v_i \in V, v_i \neq v_j\}$ is a set of edges, and $D = \vec{p}_j^{\,d}$ is a set of formation attributes that includes information such as the desired positions. Each vertex v_i has a set of neighbors $\mathcal{N}_i = \{v_j \in V \mid (i, j) \in E\}$. We also define an adjacency matrix $A_n = [a_{ij}] \in \mathbb{R}^{n \times n}$ to represent the connections between vertices. For example, if vertices v_i and v_j are connected, the corresponding entry a_{ij} in the adjacency matrix will be non-zero. An entry of the adjacency matrix is defined as

$$a_{ij} = \begin{cases} 0, & i = j \\ 0, & (i,\ j) \notin E \\ 1, & (i,\ j) \in E \end{cases} \tag{4.15}$$

The Laplacian matrix $L = [l_{ij}] \in \mathbb{R}^{n \times n}$ related to the adjacency matrix A for $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, n\}$ is defined as

$$l_{ij} = \begin{cases} \sum_{k=1}^{n} a_{ij}, & i = j \\ -a_{ij}, & i \neq j \end{cases} \tag{4.16}$$

Remark 5: Note that, in the SoNS approach, a directed edge between vertices v_j and v_i represents a communication and control link between the corresponding follower robot R_j and the leader R_i, and the indegree of each vertex is 1 (i.e., each follower has only one leader). Therefore, an n-node graph has $n-1$ edges and can be considered an n-robot formation with $n-1$ pairs of leaders and followers. For each pair, the control law Eq. (4.4) drives the follower robot to its desired position. If Theorem 4.3.2 is satisfied for all pairs, the formation of n robots will be stable (Das

et al., 2002).

Propagation of the stability bounds

As we demonstrate in Eq. (4.12), a leader-follower pair has input-to-state stability (ISS). Input-to-state stability is preserved in cascaded connections (Isidori, 1985), such that ISS bounds are calculated from one robot to another, from the first leader to the last follower of the formation. However, the upper bound of the formation's ISS depends on the initial magnitude of the formation tracking error, $\|\vec{e}_{ij}(0)\|$, and this error term tends to increase as more leader-follower pairs form complex structures. Thus, the formation's ISS depends on the longest path length of information passed from leaders to their respective followers (i.e., the length of the path from the first leader to the last follower) in the formation. Any formation can be constructed from two types of 3-robot primitives: either with cascaded connections or parallel connections. We analyze the ISS properties of these formation primitives.

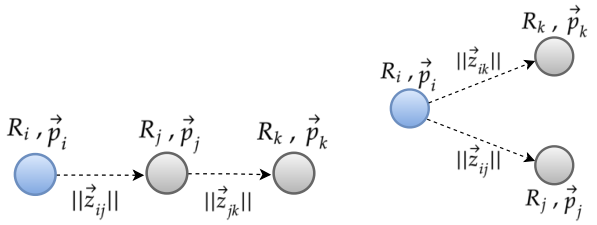


Figure 4.6: Formation primitives. (a) Cascaded formation primitive. Robot R_j follows the first leader of the formation R_i, and robot R_k in turn follows robot R_j. Dashed arrows denote information passed from leader to follower. (b) Cascaded formation primitive. Robot R_j follows the first leader of the formation R_i, and robot R_k in turn follows robot R_j. Dashed arrows denote information passed from leader to follower. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

In a formation primitive of three robots with cascaded connections (see Fig. 4.6a),

the control laws for each follower robot R_j and R_k are defined as

$$\begin{aligned} \vec{u}_j &= \boldsymbol{K}^j(\vec{z}_{ij} - \vec{d}_{ij}) = \boldsymbol{K}^j\vec{e}_{ij} \\ \vec{u}_k &= \boldsymbol{K}^k(\vec{z}_{jk} - \vec{d}_{jk}) = \boldsymbol{K}^k\vec{e}_{jk} \end{aligned} \tag{4.17}$$

The ISS bounds of each pair can be expressed as follows:

$$\begin{cases} \|\vec{e}_{ij}(t)\| & \leq \beta_{ij}(\|\vec{e}_{ij}(0)\|, t) + \gamma_{ij}\left(\sup_{0\leq\tau\leq t}\|\vec{u}_i(\tau)\|\right) \\ & \leq \left(\frac{c_2^j}{c_1^j}\right)^{\frac{1}{a_j}}\|\vec{e}_{ij}(0)\|e^{-\frac{c_3^j}{c_2^j a_j}t} + \frac{c_2^j}{c_1^j\theta}\sup_{\tau\leq t}\|\vec{u}_i(\tau)\| \\ \|\vec{e}_{jk}(t)\| & \leq \beta_{jk}(\|\vec{e}_{jk}(0)\|, t) + \gamma_{jk}\left(\sup_{0\leq\tau\leq t}\|\vec{u}_j(\tau)\|\right) \\ & \leq \left(\frac{c_2^k}{c_1^k}\right)^{\frac{1}{a_k}}\|\vec{e}_{jk}(0)\|e^{-\frac{c_3^k}{c_2^k a_k}t} + \frac{c_2^k}{c_1^k\theta}\sup_{\tau\leq t}\|\vec{u}_j(\tau)\| \end{cases} \tag{4.18}$$

The proof of *Proposition III.1* in (Tanner et al., 2004) shows that the error norm $\|\vec{e}_{jk}(t)\|$ between two followers can be expressed in terms of the first leader's velocity $\vec{u}_i$, as follows:

$$\|\vec{e}_{jk}(t)\| \leq \beta_{jk}(\|\vec{e}_{jk}(0)\|, t) + \gamma_{jk}\left(\sup_{0\leq\tau\leq t}\|\vec{u}_i(\tau)\|\right)$$

$$\beta_{jk}(\|\vec{e}_{jk}(0)\|, t) \triangleq \left(\frac{c_2^k}{c_1^k}\right)^{\frac{1}{a_k}}\|\vec{e}_{jk}(0)\|e^{-\frac{c_3^k}{c_2^k a_k}t}, \qquad \gamma_{jk}\left(\sup_{0\leq\tau\leq t}\|\vec{u}_i(\tau)\|\right) \triangleq \frac{c_2^k c_2^j}{c_1^k\theta}\sup_{\tau\leq t}\|\vec{u}_i(\tau)\| \tag{4.19}$$

To demonstrate that the ISS of the first leader-follower pair (robots R_i and R_j) can be extended to the second pair (robots R_j and R_k) in the cascaded formation primitive in Fig. 4.6a, we can construct the following composite error vector:

$$\vec{e}_{ik} \triangleq [\vec{e}_{ij} \quad \vec{e}_{jk}]^T \tag{4.20}$$

A system composed of two cascaded ISS systems is also ISS (Isidori, 1985; Khalil and Grizzle, 2002). Therefore, the composite formation tracking error of two systems

$\vec{e}_{ik}$ satisfies the inequality

$$\|\vec{e}_{ik}(t)\| \leq \beta_{ik}(\|\vec{e}_{ik}(0)\|, t) + \gamma_{ik}\left(\sup_{0\leq\tau\leq t}\|\vec{u}_i(\tau)\|\right) \tag{4.21}$$

where

$$\begin{aligned}
&\beta_{ik}(\|\vec{e}_{ik}(0)\|, t) = \beta_{ik1}(\|\vec{e}_{ik}(0)\|, t) + \beta_{ik2}(\|\vec{e}_{ik}(0)\|, t)\\
&\beta_{ik1}(\|\vec{e}_{ik}(0)\|, t) = \beta_{jk}\left(\left(2\beta_{jk}(\|\vec{e}_{ik}(0)\|, \frac{t}{2}) + \gamma_{jk}(2\beta_{ij}(\|\vec{e}_{ik}(0)\|, \frac{t}{2})) + 2\gamma_{jk}(2\beta_{ij}(\|\vec{e}_{ik}(0)\|, 0))\right), \frac{t}{2}\right)\\
&\beta_{ik2}(\|\vec{e}_{ik}(0)\|, t) = \beta_{ij}(\|\vec{e}_{ik}(0)\|, t)
\end{aligned} \tag{4.22}$$

and

$$\begin{aligned}
\gamma_{ik}\left(\sup_{0\leq\tau\leq t}\|\vec{u}_i(\tau)\|\right) = \gamma_{jk}(2\gamma_{ij}(\sup\|\vec{u}_i(\tau)\|) + 2\sup\|\vec{u}_i(\tau)\|) + \beta_{ik}(2\gamma_{jk}(2\gamma_{ij}(\sup\|\vec{u}_i(\tau)\|)\\
+ 2\sup\|\vec{u}_i(\tau)\|), 0) + \gamma_{ij}(\sup\|\vec{u}_i(\tau)\|)
\end{aligned} \tag{4.23}$$

Then, Eqs. (4.22) and (4.23) can be transformed into

$$\begin{aligned}
&\beta_{ik}(q, t) = \beta_{jk}\left(2\beta_{jk}(q, \frac{t}{2}) + 2\gamma_{jk}(2\beta_{ij}(q, 0)), \frac{t}{2}\right) + \gamma_{jk}\left(2\beta_{ij}(q, \frac{t}{2})\right) + \beta_{ij}(q, t)\\
&\gamma_{ik}(r) = \gamma_{jk}(2\gamma_{ij}(r) + 2r) + \beta_{ik}(2\gamma_{jk}(2\gamma_{ij}(r)) + 2r), 0) + \gamma_{ij}(r)
\end{aligned} \tag{4.24}$$

where $q = \|\vec{e}_{ik}(0)\|$ and $r = \sup\|\vec{u}_i(\tau)\|$.

Because a system composed of two ISS systems is also ISS (Isidori, 1985; Khalil and Grizzle, 2002), the composite formation tracking error satisfies the inequality

$$\|\vec{e}_{ijk}(t)\| \leq \beta_{ijk}(\|\vec{e}_{ijk}(0)\|, t) + \gamma_{ijk}\left(\sup_{0\leq\tau\leq t}\|\vec{u}_i(\tau)\|\right) \tag{4.25}$$

where

$$\begin{aligned}\beta_{ijk}(\|\vec{e}_{ijk}(0)\|, t) &= \beta_{ij}(\|\vec{e}_{ij}(0)\|, t) + \beta_{ik}(\|\vec{e}_{ik}(0)\|, t) \\ \gamma_{ijk}\left(\sup_{0\leq\tau\leq t}\|\vec{u}_i(\tau)\|\right) &= \gamma_{ij}(\sup\|\vec{u}_i(\tau)\|) + \gamma_{ik}(\sup\|\vec{u}_i(\tau)\|)\end{aligned} \tag{4.26}$$

Then, Eq. (4.26) can be rewritten as

$$\begin{aligned}\beta_{ijk}(q, t) &= \beta_{ij}(q, t) + \beta_{ik}(q, t) \\ \gamma_{ijk}(r) &= \gamma_{ij}(r) + \gamma_{ik}(r)\end{aligned} \tag{4.27}$$

In a formation primitive of three robots with parallel connections (see Fig. 4.6b), both follower robots R_j and R_k can be assumed to be equivalent to the first follower robot R_j in the cascaded formation primitive shown in Fig. 4.6a.

Error norm between the first leader and the first follower

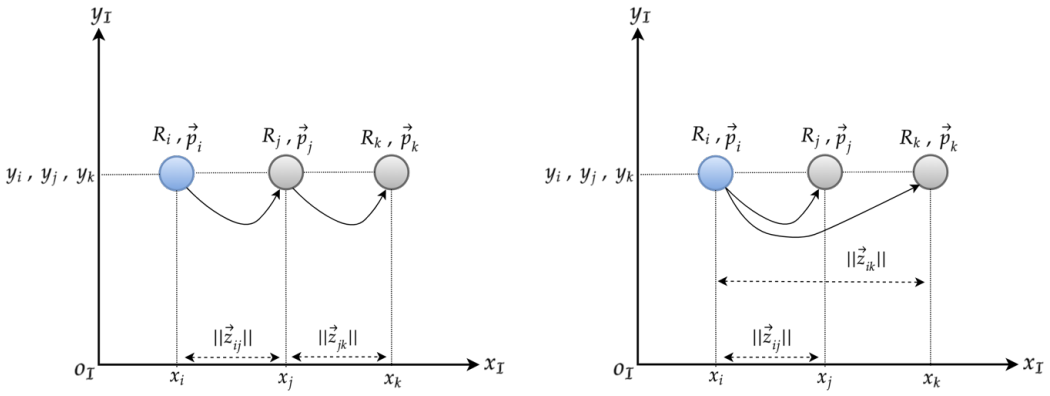


Figure 4.7: Two example formations of three robots. The arrows denote graph connections, where information is passed from leader to follower. (a) Cascaded connections. (b) Parallel connections. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

We assume that both formations move along the y-axis. The closed-loop kine-

matics for the connections can be given as

$$\begin{aligned}\vec{u}_j &= \boldsymbol{K}^j(\vec{z}_{ij} - \vec{d}_{ij}) \\ \vec{u}_k &= \boldsymbol{K}^k(\vec{z}_{jk} - \vec{d}_{jk})\end{aligned} \tag{4.28}$$

where $\boldsymbol{K}^j$ and $\boldsymbol{K}^k$ are controller gain matrices and $k_1^j = k_2^j = k_1^k = k_2^k = 5$. Note that the initial positions of the robots are $\vec{p}_i(0) = [5,\ 10]^T$, $\vec{p}_j(0) = [1,\ 3]^T$, and $\vec{p}_j(0) = [3,\ -2]^T$. We set the constant reference velocity and desired displacement vectors as $\vec{u}_i = [0,\ 10]^T$, $\vec{d}_{ij} = [6,\ 0]^T$, $\vec{d}_{jk} = [6,\ 0]^T$, $\vec{d}_{ik} = [12,\ 0]^T$.

To calculate the formation ISS measure P_{ISS} for the cascaded connections (shown in Fig. 4.7a), we redefine the asymptotic ISS gain Eq. (4.24) by setting $r = 1$, $q = 1$, and $t = 0$. This bounds the inputs of the first leader inside the unit ball and ensures that P_{ISS} varies in the range of $[0,\ 1]$. This results in the following expression:

$$\begin{aligned}\gamma_{ik}(1) &= \gamma_{jk}(2\gamma_{ij}(1) + 2) + \beta_{ik}(2\gamma_{jk}(2\gamma_{ij}(1) + 2), 0) + \gamma_{ij}(1) \\ &= 2\hat{\gamma}_{jk}\hat{\gamma}_{ij} + 2\hat{\gamma}_{jk} + 4\hat{\beta}_{jk}\hat{\gamma}_{jk}\hat{\gamma}_{ij} + 4\hat{\beta}_{jk}\hat{\gamma}_{jk} + \hat{\gamma}_{ij}\end{aligned} \tag{4.29}$$

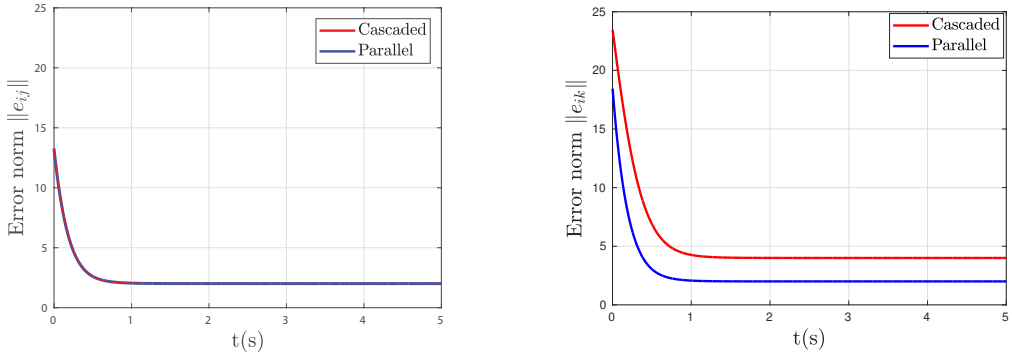
where $\hat{\beta}_{ij} \triangleq \left(\frac{c_2^j}{c_1^j}\right)^{\frac{1}{a_j}}$, $\hat{\beta}_{jk} \triangleq \left(\frac{c_2^k}{c_1^k}\right)^{\frac{1}{a_k}}$, $\hat{\gamma}_{ij} \triangleq \frac{c_2^j}{c_1^j\theta}$, $\hat{\gamma}_{jk} \triangleq \frac{c_2^k}{c_1^k\theta}$, $c_1^j \triangleq \min(k_1^j, k_2^j)$, $c_2^j \triangleq \max(k_1^j, k_2^j)$, $c_1^k \triangleq \min(k_1^k, k_2^k)$, $c_2^k \triangleq \max(k_1^j, k_2^j)$, and $a_j = a_k = 2$. This results in the following P_{ISS} value:

$$P_{ISS}^c = \frac{1}{1 + \gamma_{ik}(1)} = \frac{\theta^2}{\theta^2 + 7\theta + 6} \tag{4.30}$$

where $\theta \in (0,\ 1)$.

The follower robots R_j and R_k with parallel connections (see Fig. 4.6b) can be assumed to be equivalent to the first follower robot R_j with cascaded connections (see Fig. 4.6a). A follower that is directly connected to the first leader has a lower magnitude of relative errors with respect to the first leader's velocity than a follower that is instead connected to another follower. Fig. 4.8a shows a comparison of the error norm for the first leader and its first follower in the two formations. In both formations, R_j is in the first hierarchy layer and therefore the error norms $\|e_{ij}\|$ for

R_j are the same. However, in the cascaded connections, robot R_k follows R_j, while in the parallel connections, R_k directly follows R_i. As a result, the error norm $\|e_{ik}\|$ for R_k in the cascaded connections is higher than that for all other followers in both formations.



(a) (b)

Figure 4.8: Comparison of two formations' tracking error evolutions. (a) Error norm between the first leader R_i and first follower R_j. Note that the lines for the cascaded and parallel cases are the same (i.e., the red and blue lines of the graph are in the same position, and therefore not fully visible). (b) Error norm between the first leader R_i and last robot R_k. (Reprinted from the supplementary materials of (Zhu et al., 2024).)

ISS gains calculation for n-robot formations

To obtain the total ISS gains for a formation of multiple leader–follower pairs, we start from the terminal nodes (i.e., vertices with outdegree of 0) and apply Eqs. (4.22), (4.23), and (4.26) iteratively based on the algorithm given in (Tanner et al., 2004), thus reducing the original graph to a depth of one.

To illustrate this calculation for a specific target topology, consider a graph $G = (V, E, D)$ with an $n \times n$ adjacency matrix A, where a_i represents the i-th row. Begin

by defining the following vectors:

$$\hat{\beta}^0 \triangleq [\hat{\beta}_1{}^0 \dots \hat{\beta}_n{}^0]^T \tag{4.31}$$

$$\hat{\gamma}^0 \triangleq [\hat{\gamma}_1{}^0 \dots \hat{\gamma}_n{}^0]^T \tag{4.32}$$

where $\hat{\beta}$ and $\hat{\gamma}$ are the gain estimates defined in Eq. (4.11).

After $k+1$ iterations, we obtain:

$$\begin{aligned} \hat{\beta}^{k+1} &\triangleq [\hat{\beta}_1{}^{k+1} \dots \hat{\beta}_n{}^{k+1}]^T \\ \hat{\gamma}^{k+1} &\triangleq [\hat{\gamma}_1{}^{k+1} \dots \hat{\gamma}_n{}^{k+1}]^T \end{aligned} \tag{4.33}$$

Then, $\hat{\beta}^{k+1}$ and $\hat{\gamma}^{k+1}$ can be calculated as

$$\begin{aligned} \hat{\beta}^{k+1} &= \hat{\beta}^k + \eta_{n-k} c_\beta^k \\ \hat{\gamma}^{k+1} &= \hat{\gamma}^k + \eta_{n-k} c_\gamma^k \\ \eta_{n-k} &= [\underbrace{0 \ \dots \ 0}_{n-k-1} \ 1 \ \dots \ 0]^T \\ c_\beta^k &= a_{n-k}\hat{\beta}^k a_{n-k}\hat{\gamma}^k \hat{\beta}_{n-k}^k + (a_{n-k}\hat{\beta}^k)^2 \\ c_\gamma^k &= a_{n-k}\hat{\beta}^k a_{n-k}\hat{\gamma}^k \hat{\gamma}_{n-k}^k + a_{n-k}\hat{\gamma}^k \hat{\gamma}_{n-k}^k \end{aligned} \tag{4.34}$$

For any formation, the algorithm that is iteratively applied, Eqs. (4.22), (4.23), and (4.26), will terminate in at most $n-1$ steps (i.e., the maximum path length in a graph with n vertices).

It is important to note that the depth of a formation's graph will affect its stability: the higher the depth of the graph, the larger the ISS gains will be. Similarly, the depth of the graph will affect its robustness (Wang et al., 2009) in response to noisy local information (i.e., under random disturbances in the information transferred between robots).

4.3.4 Incorporating feed-forward information

If a follower robot receives velocity information (feed-forward) from its leader without any time delay, the control law Eq. (4.3) can be rewritten as follows:

$$\begin{aligned} \vec{u}_j &= f(t, \vec{z}_{ij}, \vec{d}_{ij}) + \vec{u}_i \\ &= \boldsymbol{K}^j(\vec{z}_{ij} - \vec{d}_{ij}) + \vec{u}_i \\ &= \boldsymbol{K}^j \vec{e}_{ij} + \vec{u}_i \end{aligned} \tag{4.35}$$

where the first term in the control input is used to maintain the desired displacement from the leader and the second term is used to follow the leader's reference velocity.

If we substitute the control law given by Eq. (4.35) into the equation for the motion model given by Eq. (4.1), we obtain

$$\dot{\vec{p}}_j = \boldsymbol{K}^j \vec{e}_{ij} + \vec{u}_i \tag{4.36}$$

Then, the leader-follower kinematics given by Eq. (4.2) can be rewritten as

$$\dot{\vec{e}}_{ij} = -\boldsymbol{K}^j \vec{e}_{ij} \tag{4.37}$$

By solving the differential equation Eq. (4.37), we then obtain

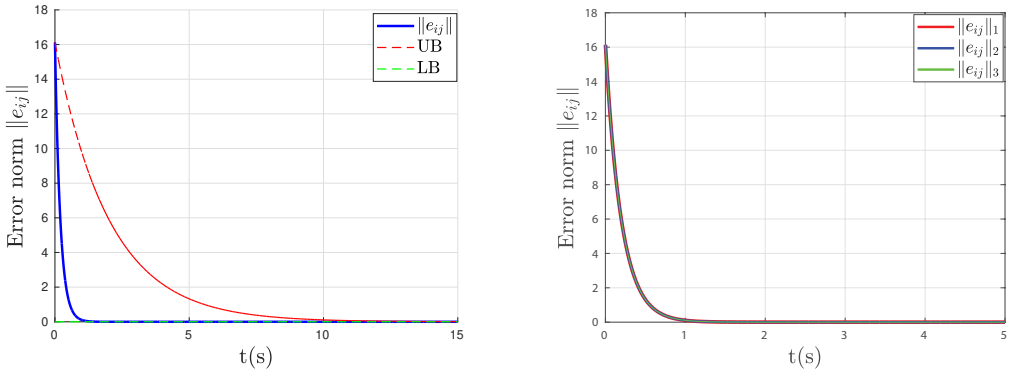
$$\vec{e}_{ij}(t) = \mathcal{C} e^{-(\boldsymbol{K}^j)^T t} \tag{4.38}$$

where $\mathcal{C}$ is any positive constant. Since $(\boldsymbol{K}^j)^T = \boldsymbol{K}^j$ is positive definite, then $\|\vec{e}_{ij}(t)\| \to 0$. As $t \to \infty$, $\dot{\vec{p}}_j$ (or $\vec{u}_j$) approaches $\vec{u}_i$, such that the leader can steer the follower with the velocity $\vec{u}_i$.

Remark 6: Unlike the control law given by Eq. (4.4), the control law in Eq. (4.35) is not influenced by the leader's velocity, due to the use of feed-forward information. This means that the stability of the leader-follower pair is not affected by whether the leader is stationary or moving.

To demonstrate the benefit of using feed-forward information, we conduct a simulation using the same parameters as the previous case, but with the follower robot re-

ceiving the leader's velocity information without any delay. The results in Fig. 4.9a,b, demonstrate that the error norm indeed approaches zero regardless of the leader's velocity.



(a) (b)

Figure 4.9: Norm of the relative formation tracking error with feed-forward information. (a) Stationary leader with the upper bound (UB) and lower bound (LB) of the error norm. Note that the lower bound line (green) is along the bottom of the graph. (b) Moving leader under different velocity regimes. Note that the lines for the three velocity regimes are the same (i.e., the red, blue, and green lines of the graph are all in the same position, and therefore not fully visible). (Reprinted from the supplementary materials of (Zhu et al., 2024).)

4.4 Results

We demonstrate the capabilities of the SoNS approach in experiments using real heterogeneous aerial-ground swarms consisting of standard differential drive e-puck robots (Millard et al., 2017; Mondada et al., 2009) and our custom-developed S-drone quadrotors (see Chapter 3 for the details). To also demonstrate the SoNS capabilities in swarms larger than our real robot arena allows, we run experiments in a simulator that is cross-verified against the behavior of the real robots (using ARGoS (Pinciroli et al., 2012). Within each experiment, all robots run local copies of the same SoNS algorithm and operate fully autonomously—without any global positioning system,

remote control station, or off-board sensing. The robots use vision-based relative positioning and are allowed to communicate wirelessly only if one robot is in the other's field of view. Actuation in the experiments is confined to motion.

4.4.1 Analysis metrics

For our empirical evaluation, we quantify each robot's deviation from its eventual settled pose by defining an actuation error ϵ_{r_i} for robot r_i. Denote the current time by t, the completion time of the present target graph by $\textsc{Final}$, the pose of robot r_i at time t by $\boldsymbol{s}_{r_i}^t$, its destined pose at time $\textsc{Final}$ by $\boldsymbol{s}_{r_i}^{*\textsc{Final}}$, the brain's pose at time t by $\boldsymbol{s}_{\textsc{Brain}}^t$, and the brain's pose at time $\textsc{Final}$ by $\boldsymbol{s}_{\textsc{Brain}}^{\textsc{Final}}$. Using $\textsc{disp}(x, y)$ for the displacement of x relative to y in y's body frame and the Euclidean norm $||\cdot||$, we compute:

$$\epsilon_{r_i}^t = \left\| \textsc{disp}\left(\boldsymbol{s}_{r_i}^t, \boldsymbol{s}_{\textsc{Brain}}^t\right) - \textsc{disp}\left(\boldsymbol{s}_{r_i}^{*\textsc{Final}}, \boldsymbol{s}_{\textsc{Brain}}^{\textsc{Final}}\right) \right\| \tag{4.39}$$

and the swarm's mean actuation error at time t as

$$E^t = \frac{1}{n} \sum_{i=1}^{n} \epsilon_{r_i}^t \tag{4.40}$$

where n is the number of robots.

To benchmark performance, we introduce a lower bound B^t on the mean error, representing the scenario in which each robot travels straight to its target at maximum speed along the shortest path (ignoring obstacles, reconfiguration overhead, and inter-robot collisions), and adjusting for aerial hover error. First, for each robot:

$$b_{r_i}^t = \left\| \textsc{disp}\left(\boldsymbol{s}_{r_i}^{\textsc{Start}}, \boldsymbol{s}_{\textsc{Brain}}^{\textsc{Start}}\right) - \textsc{disp}\left(\boldsymbol{s}_{r_i}^{*\textsc{Start}}, \boldsymbol{s}_{\textsc{Brain}}^{\textsc{Start}}\right) \right\| - \kappa_i \left(t - \textsc{Start}\right) \tag{4.41}$$

and then the swarm's lower bound:

$$B^t = \begin{cases} \dfrac{1}{n} \displaystyle\sum_{i=1}^{n} b_{r_i}^t \; - \; 2h, & \text{if } \; \frac{1}{n} \sum b_{r_i}^t > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4.42}$$

Here, START marks the beginning of the current target graph, $\boldsymbol{s}_{r_i}^{\text{START}}$ and $\boldsymbol{s}_{r_i}^{*\text{START}}$ are the robot's actual and intended poses relative to $\boldsymbol{s}_{\text{BRAIN}}^{\text{START}}$, κ_i is robot r_i's maximum speed, and h is the mean hover error for aerial units (the $2h$ term compensates for possible mutual drift that could artificially shrink pairwise distances). Although each $b_{r_i}^t$ decreases linearly at rate κ_i, the aggregate B^t often tapers nonlinearly as robots begin from varied displacements.

4.4.2 Robot missions

We conduct proof-of-concept robot missions that together demonstrate the key capabilities and novel features of the SoNS approach. All missions demonstrate self-organized controllable hierarchy, the first feature of the SoNS approach: hierarchy was maintained despite external disturbances. The novel features of interchangeable leadership and explicit inter-system reconfiguration are both demonstrated in the splitting and merging mission, as robots reconfigure into different SoNSs and also reconfigure their leadership allocations during the splits and merges. Note that interchangeable leadership is also demonstrated in the later fault-tolerance experiments. Together, these missions demonstrate the ability to self-organize multi-level system architectures, including their communication structures, inter-level control distributions, and system behaviors.

For each mission, we report at least five trials with real robots and 50 trials in simulation (with up to 65 robots), with a maximum run time of 15 minutes (constrained by the battery capacity of the quadrotor platform). The goals and scope of possible behaviors for each mission were designed offline. We give the mission schematics, show that the qualitative goals of the mission were achieved, and assess the results in terms of mean actuation error E with respect to a lower bound B. The mean error E with respect to lower bound B is meant to be a comprehensive metric that encapsulates all types of actuation error that can originate in the SoNS, such as errors in sensing, in decision making, during reconfiguration, or while converging on a single shared SoNS.

Establishing self-organized hierarchy

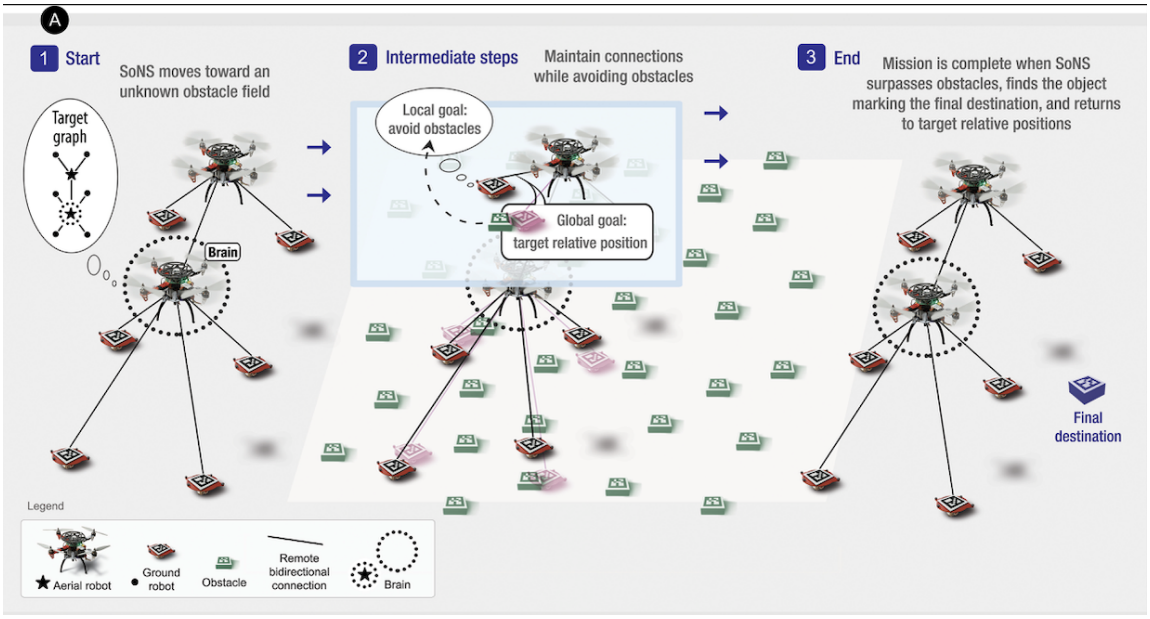


Figure 4.10: Establishing self-organized hierarchy. In this experiment, independent robots in arbitrary starting positions need to self-organize into a single SoNS with the target multi-level communication structure and target relative poses. Mission schematic: (1) Starting on the ground, some robots are positioned in tight clusters, which increases the possibility for interference between robots. (2) To self-organize into the target graph, first the aerial robots take off and all robots begin searching for peers. Then, robots start forming connections, merging their respective SoNSs, and reallocating themselves into positions that match the target graph. During this process, robots continually adjust their relative positions while coordinating locally to avoid collisions, until (3) the target graph is complete *(figure continued on next page)*. (Reprinted from (Zhu et al., 2024).)

The first proof-of-concept mission demonstrates the establishment of SoNSs (see Fig. 4.10): independent robots in arbitrary start positions need to self-organize into a single hierarchical system with the target multi-level communication structure, including the correct topology and correct relative positions. In the two mission variants, robots start on the ground either (i) with some of the robots clustered tightly (and therefore resulting in robot–robot interference) in varying arbitrary locations, or

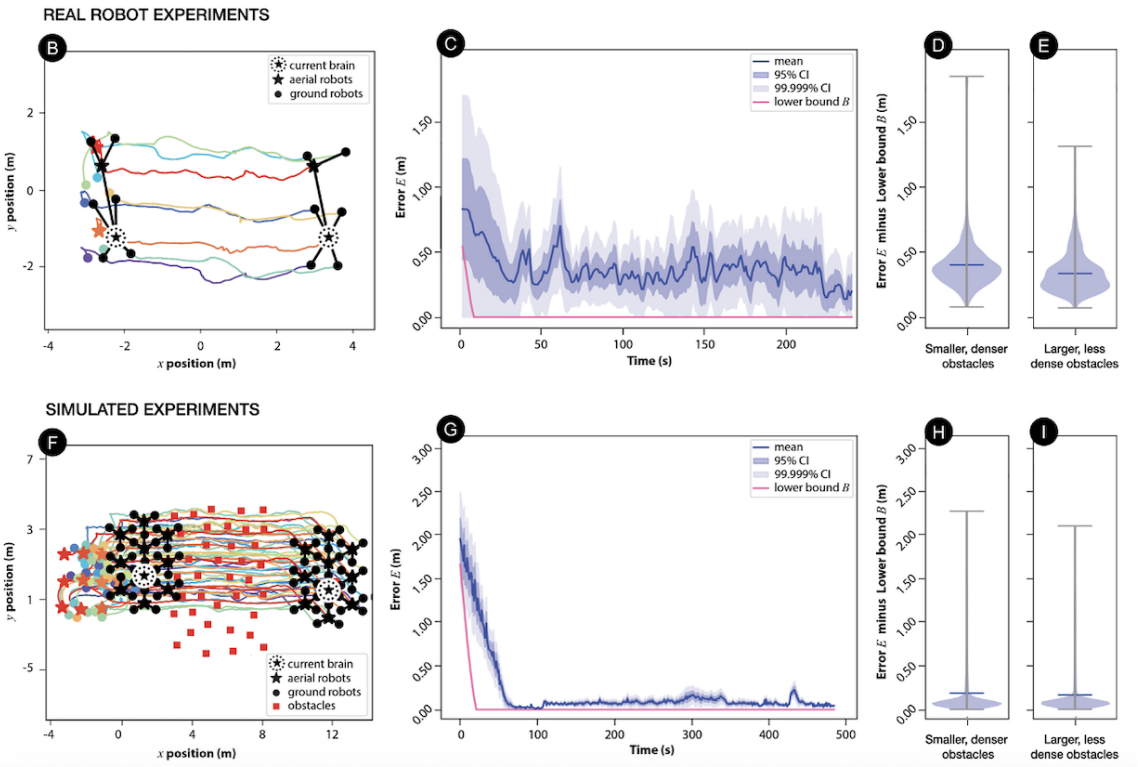


Figure 4.10: *(cont'd)* (B-E) Results of real robot experiments. (B) Trajectories of robots over time, in a trial with 12 robots, with the initial and final positions indicated in color and in black, respectively, and (C) actuation error E (mean and confidence interval per robot over time, see Eq. 4.40, with lower bound B, see Eq. 4.42, plotted for reference) in an example trial. (D,E) Violin plots of $E - B$, actuation error minus lower bound, in all experiment trials, for both mission variants: (D) clustered start, six trials, and (E) scattered start, five trials. In this and following figures, the violin plots show that the given example trial is not an outlier and is indicative of the overall results of the respective mission. (Reprinted from (Zhu et al., 2024).)

(ii) with all robots scattered in varying arbitrary positions throughout the arena. All robots run local copies of the same SoNS algorithm and begin as independent single-robot SoNSs, of which they are the brain by default. As robots enter each other's FoV, they compete to recruit each other, with many merging operations happening simultaneously. Meanwhile, robots with children also continuously (re)allocate them based on their target graphs. Thus, the robots self-organize not only to converge on a single SoNS but also to approach the target relative poses indicated by the attributes of the brain's target graph.

In all experiments, the robots complete the mission: they converge on one SoNS and establish the correct topology and relative positions for the target communication topology (see example trial in Fig. 4.10B). The progression of a typical experiment can be seen in the example trial (see Fig. 4.10C), in which the robots converge to the correct topology and reach a low steady-state actuation error. All trials reach a low steady-state error, with the clustered start variant displaying slightly higher average error than the scattered start variant (see Figs. 4.10D,E).

Splitting and merging

The last proof-of-concept setup with real robots is a search-and-rescue mission (see Fig. 4.11), in which SoNSs split and merge in order to reunite with missing robots. At initiation, one or more single-robot SoNSs are isolated somewhere in the environment and wait there to be found by a rescue team. There is also a primary multi-robot SoNS somewhere in the environment. This SoNS notices that it is incomplete and starts a rescue mission to find any missing robots (see Fig. 4.11A). The brain issues instructions downstream for a specific node in its target graph to split away and activate the 'rescuer' program. When the robot at that node receives the message, it splits from its parent and automatically becomes a brain. It also lowers its quality, so that it can resume its old membership upon returning. When robots in the newly split 'rescuer' SoNS detect objects. The brain of this new SoNS uses sensor information about these objects to follow them until it recruits a new robot, and therefore acquires the correct number of children for its target graph. It then uses

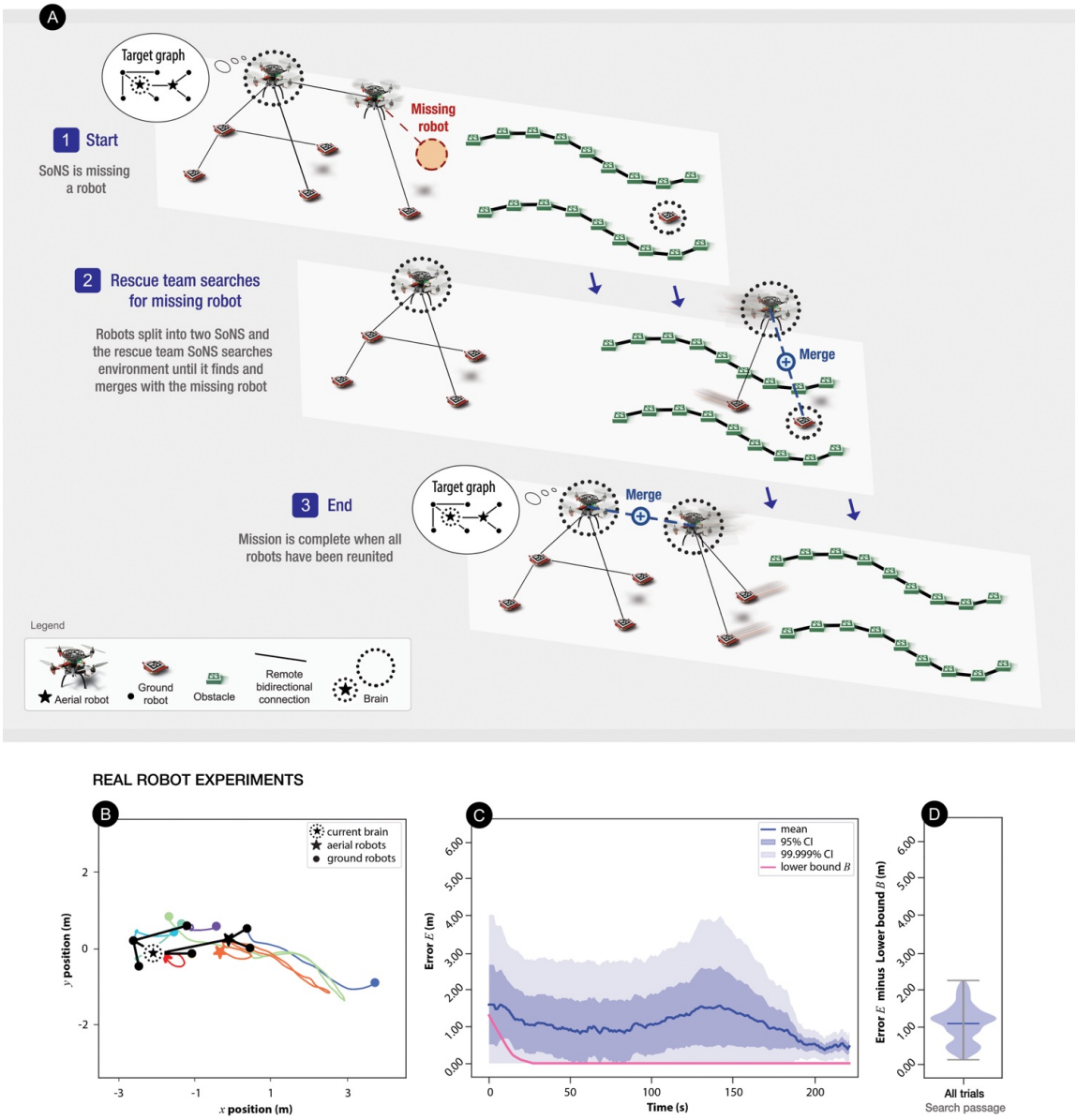


Figure 4.11: Splitting and merging. In this experiment, robots needed to conduct multi-SoNS search-and-rescue to reunite with missing robot(s). {(A) Mission schematic: (1) Robots started in a SoNS that was missing a member. (2) The brain instructed one of the robots to split and temporarily form its own multi-robot SoNS as a rescue team. The rescue team SoNS searched a passage in the environment until it found the missing robot and merged with it. (3) The rescue team SoNS returned to the initial split location and re-merged with the remaining SoNS, reuniting all robots. (B-D) Results of real robot experiments, in trials with 8 robots. (B,C) In a real robot example trial: (B) trajectories of robots over time, with the initial (in color) and final (in black) positions indicated, and (C) mean actuation error E (Eq. 4.40) with lower bound B (Eq. 4.42) plotted for reference. (D) Violin plot of E (Eq. 4.40) minus B (Eq. 4.42) in all five trials. (Reprinted from (Zhu et al., 2024).)

sensor information to backtrack along the same objects. When it encounter its former SoNS, it gets re-recruited, and as a result the two SoNS re-merge.

In all trials, the robots successfully complete the mission goals. The actuation error rises during the period of reorganizations, but all robots re-merge into one SoNS and the system returns to a low steady-state error (see Fig. 4.11C,D).

This mission includes three different variants, two run in experiments with real robots and one run in simulation (see Figs. 4.12–4.16) In an alternative mission variant, the primary SoNS knows the direction of its missing robot, but needs to physically push an obstruction out of the way, then merge with the missing robot and guide it out of a convex barrier.

Fig. 4.12 shows a representative split-and-merge cycle during a search-and-rescue mission in an obstacle-filled arena. In panel (a) the full SoNS is assembled but one or two robots remain missing (greyed-out). Panel (b) captures the moment when the SoNS "brain" issues a split command, detaching a small sub-team to form an independent rescue group. Panels (c) and (d) depict how this rescue sub-team fans out around obstacles, performs a systematic corridor sweep, locates the missing robot, and immediately merges their local maps and beliefs. Finally, panels (e) and (f) illustrate the return journey: the augmented rescue SoNS retraces its path through the passage and rejoins the original group, completing the reunification.

Fig. 4.13 summarizes the quantitative results from five independent real-robot trials, each with eight ground robots. In each row, the left plot shows the overhead trajectory log for one run (with colored lines for individual robot paths, while the right plot shows the error to the missing robot over time. Across all five trials the rescue sub-team departs correctly, reduces error sharply upon locating its target, and completes the final merge in under 90 s, converging to within 0.5 m of the missing robot with low variance. The split phase consistently delivers rapid coverage, driving average error below 3 m within 20–30 s, and the subsequent merge phase refines localization to sub-meter accuracy. Despite real-world disturbances—wheel slip, sensor noise, intermittent communications—the split-merge strategy reliably reunites all robots with steady-state error below 1 m, demonstrating the robustness of the SoNS framework in cluttered environments.

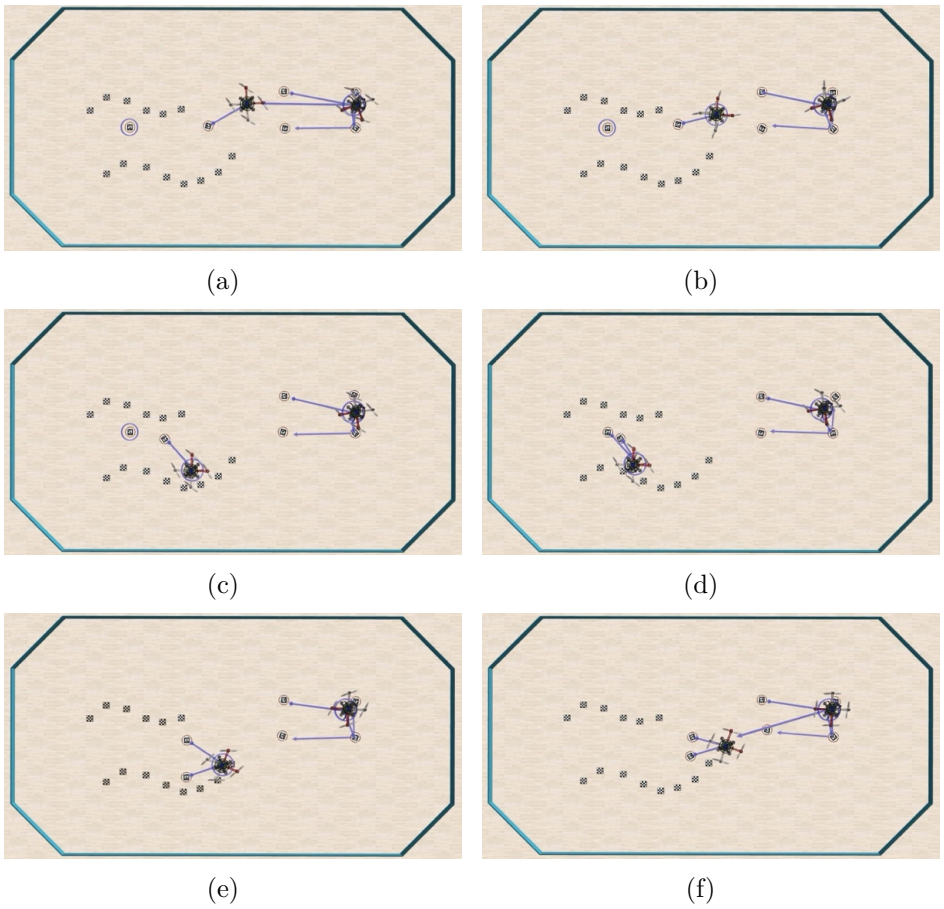


Figure 4.12: Splitting and merging systems, search and rescue: Key frames. (a) When robots start, they are in a SoNS that is missing one or two of its members. (b) The SoNS-brain instructs one of the robots to split from it and temporarily form its own multi-robot SoNS as a rescue team. (c,d) The rescue team SoNS searches the environment until it finds the missing robot(s) and merges with it/them. (e,f) The merged rescue team SoNS then returns to the location where it initially split off and re-merges with the remaining SoNS, so that all robots are reunited, and the mission is complete. Note that, in the experiments, there are more obstacles in the environment, forming the walls of an arbitrary passage that the robots need to navigate to complete the search and rescue mission. (Reprinted from (Zhu et al., 2024)).

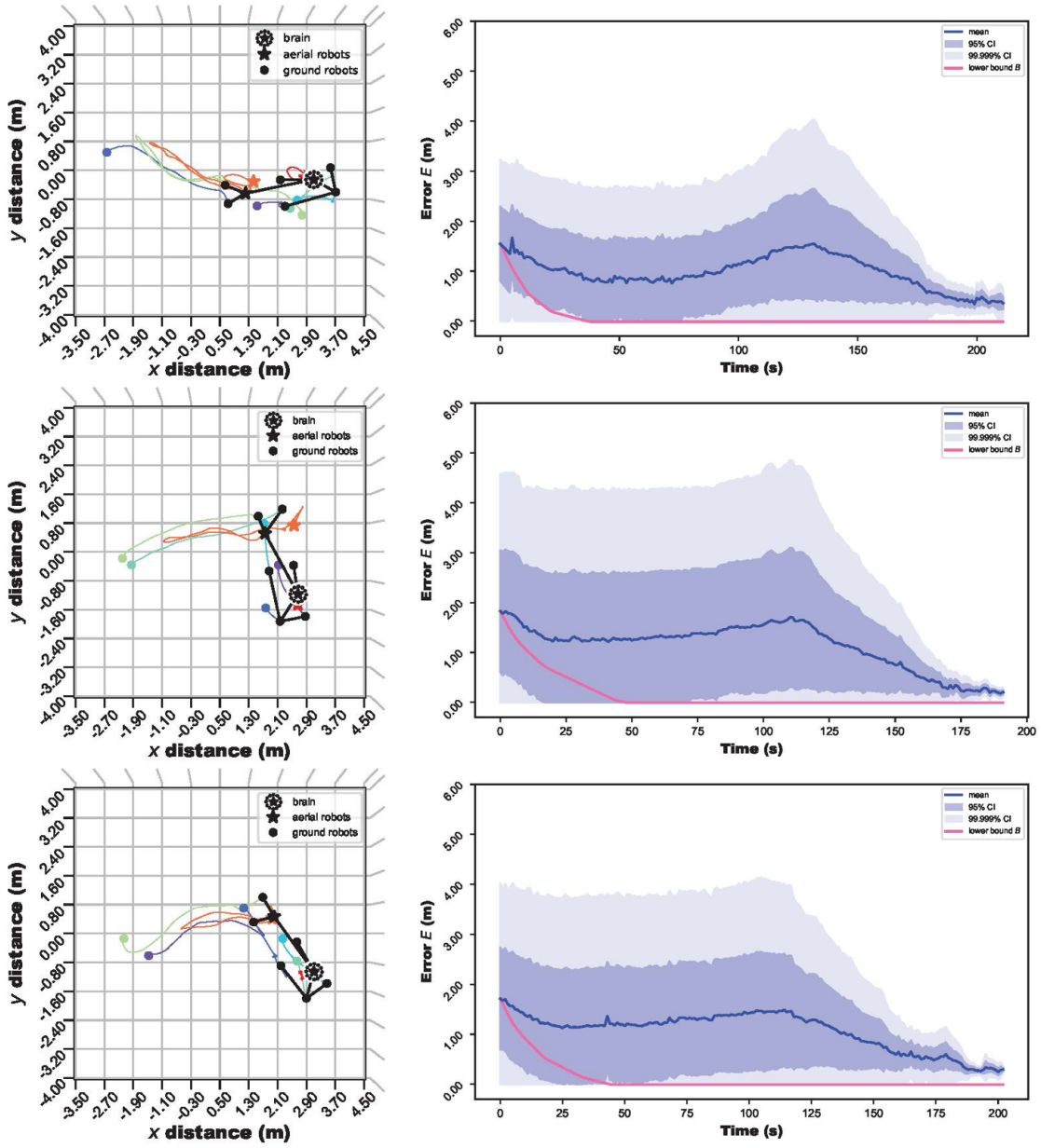


Figure 4.13: Splitting and merging systems, search and rescue: Real robot trials. Five trials with real robots were conducted, each with eight robots *(figure continued on next page)*. (Reprinted from (Zhu et al., 2024).)

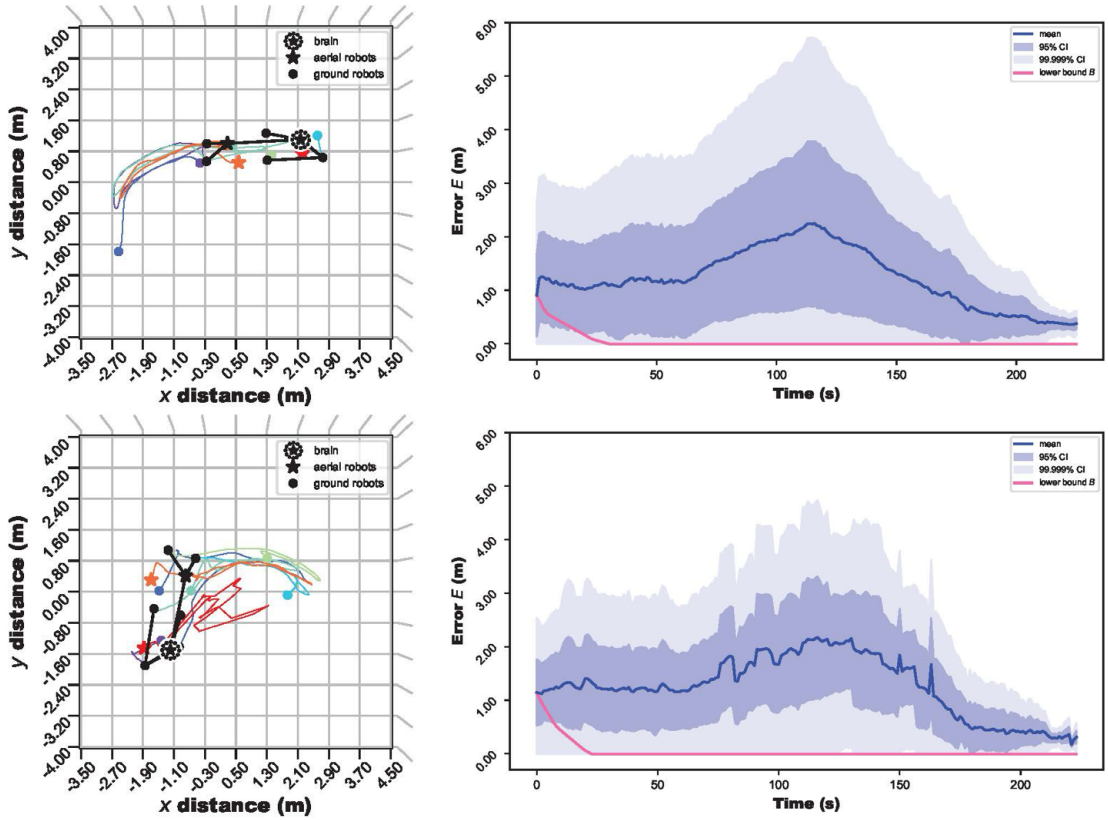


Figure 4.13: *(cont'd)* Splitting and merging systems, search and rescue: Real robot trials. Five trials with real robots were conducted, each with eight robots. (Reprinted from (Zhu et al., 2024).)

Figure 4.14 presents a sequence of six key frames from a single push-away mission in an obstacle-cluttered arena. In panel (a) the full SoNS is assembled but one robot remains trapped behind a barrier. Panel (b) shows how the SoNS "brain" directs the free units to search until they locate the missing robot. Panel (c) captures the instant when the team reorganizes into a multi-robot pushing formation around the obstruction. Panels (d) and (e) illustrate the collaborative push as the ground robots lever the barrier out of the way, allowing the missing robot to rejoin. Finally, panel (f) depicts the reconfigured SoNS adopting its target topology once the obstruction has been cleared and all eight robots are reunited.

Figure 4.15 summarizes five independent real-robot trials of the push-away variant. In each row the left-hand plot shows the overhead trajectory of all eight robots (colored lines), with black markers indicating the push formation point, a red star marking the location of the trapped robot, and filled squares at the final merge. The corresponding right-hand plot traces the error between the free SoNS and the missing robot over time. All runs exhibit an initial error of 1–2m during the search phase, a transient rise to 2–2.5m when the obstruction is encountered, and a sharp decline below 0.5m as the team completes the push and merges. Steady-state error remains under 0.6 m with narrow confidence bands, despite wheel slip and sensor drift.

The push-away strategy thus achieves reliable reunification and obstacle removal in under 90s on average. During the search phase the SoNS locates the trapped robot in 20–40s, then transitions smoothly into a pushing formation that reduces localization error by over 75% in under 15s. The final merge with the rescued robot consistently brings residual error below 0.5m, demonstrating that the SoNS framework can coordinate both exploratory search and heavy-load manipulation under real-world disturbances.

Figure 4.16 shows three representative simulation trials of the simple split-and-merge variant executed 50 times with 50 robots. In each row the left panel depicts the overhead trajectories: robots depart from an initial lattice, execute a split at the central waypoint (black circle), traverse the corridor in two sub-teams, and reform at the merge point (filled square). Colored traces indicate individual robot paths, illustrating uniform coverage and collision avoidance across all agents. The right

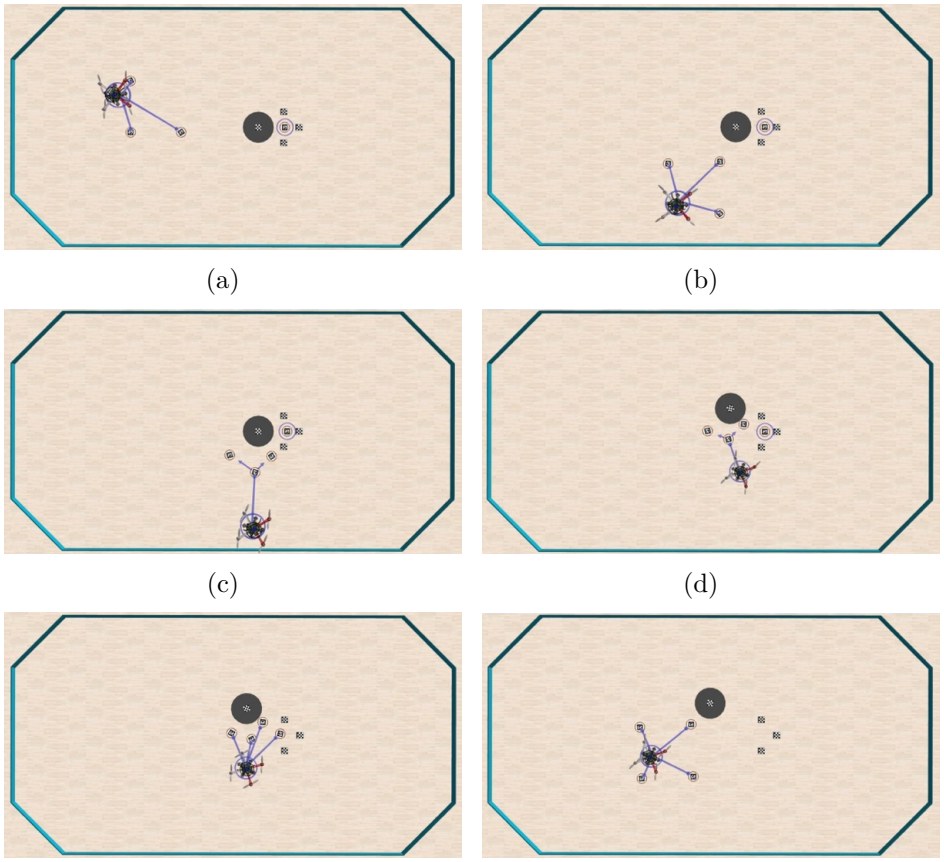


Figure 4.14: Push away an obstruction: Key frames. (a) When robots start, they are in a SoNS that is missing a member. (b) The SoNS searches the environment and finds the missing robot trapped by obstacles. (c) The SoNS reorganizes into a shape that allows its ground robots to collaboratively push a large obstruction. (d,e) The robots push the obstruction away and the SoNS merges with the missing robot, so that all robots are reunited. (f) The robots reorganize into the target SoNS and the mission is complete. (Reprinted from (Zhu et al., 2024).)

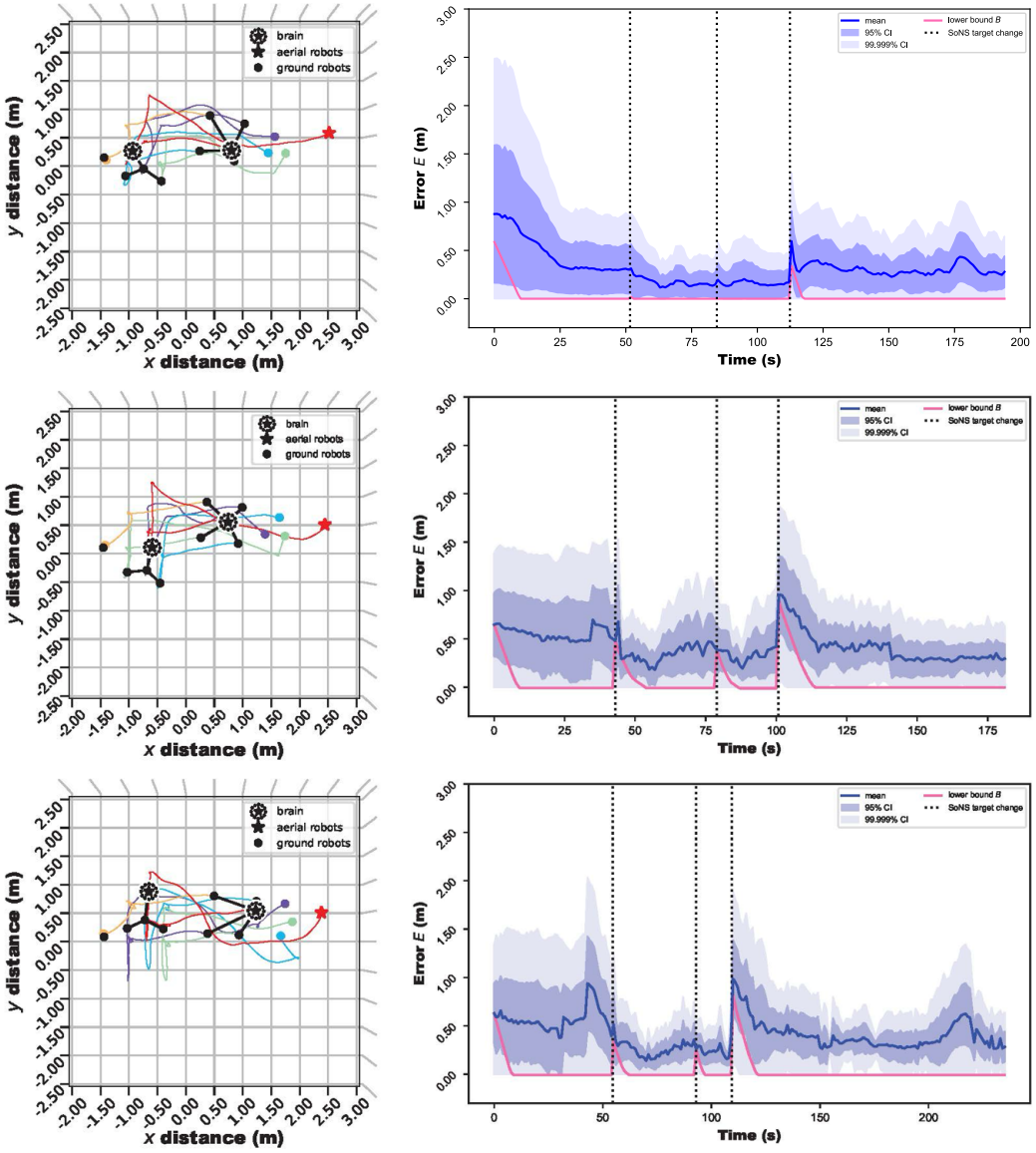


Figure 4.15: Splitting and merging systems, push away an obstruction: Real robot trials. Note that the highlighted keyframes shown here are from an intermediary time step, not the end of the trial. Five trials with real robots were conducted, each with eight robots *(figure continued on next page)*. (Reprinted from (Zhu et al., 2024).)

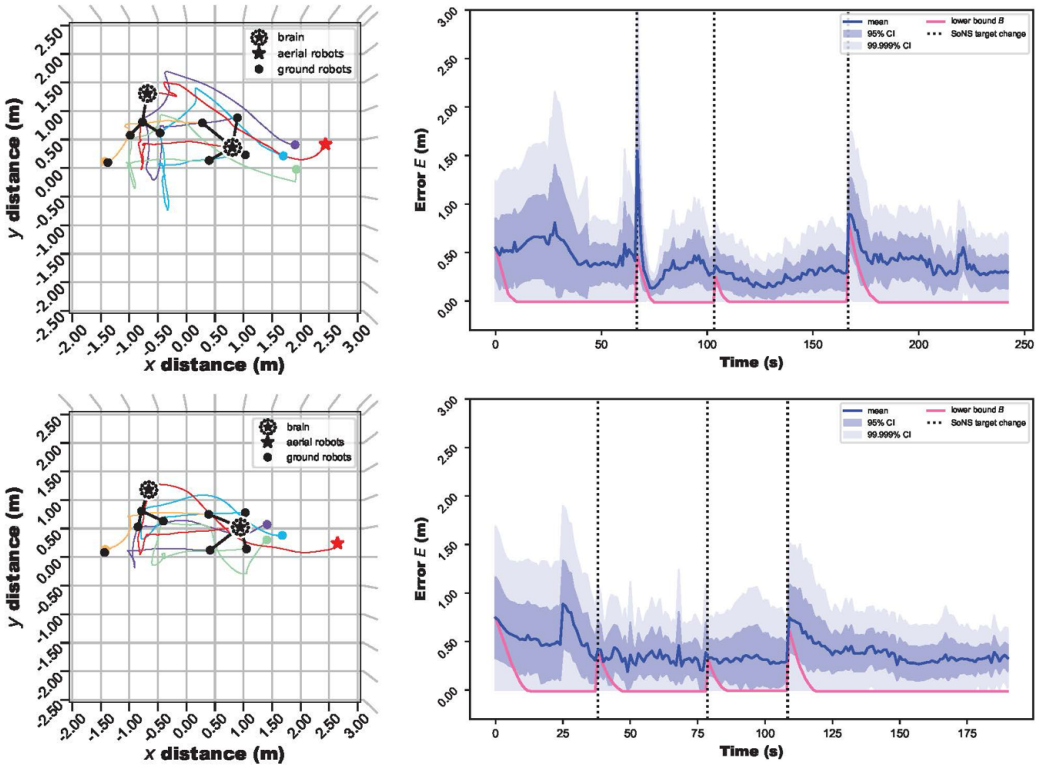


Figure 4.15: *(cont'd)* Splitting and merging systems, push away an obstruction: Real robot trials. Note that the highlighted keyframes shown here are from an intermediary time step, not the end of the trial. Five trials with real robots were conducted, each with eight robots. (Reprinted from (Zhu et al., 2024).)

panel plots the error between the split sub-team's centroid and the mission waypoint over time. The solid blue line is the mean error across all 50 trials, the dark and light shaded regions correspond to the 95% and 90% confidence intervals, respectively. The split phase drives error from roughly 1.2m to below 0.3m within 30s, after which error remains near zero during traversal. A transient peak of 0.7–1.0m appears around 150–180s as the sub-teams merge, before the error converges back to under 0.2m in the final steady-state. The consistently narrow confidence intervals demonstrate that, even without search-and-rescue subtasks, the simple split-and-merge primitives scale robustly to large teams under simulated sensor noise and communication delays, achieving reliable reunification with sub-meter accuracy in under three minutes.

We demonstrate several aspects of fault tolerance in SoNS, in real and simulated swarms. First, using real robots, we show replacement of a single robot that has failed permanently, including a failed brain. In these demonstrations, one robot was remotely triggered to fail (for the aerial robots, this included immediately landing in place). Then, a new robot of the same hardware type was manually placed in the arena and switched on, after which it was recruited by the SoNS. When a brain robot or a robot at an inner hierarchy level failed, it was immediately and automatically replaced by another robot already in the SoNS, and the robots redistributed around the change. Then, when a new robot was recruited, it filled the vacancy in the reorganized SoNS.

Using real robots, we also demonstrate reorganization in high-loss conditions (Fig. 4.17A-C), i.e., after arbitrary permanent failures when the failed robots cannot be replaced, with five trials run. The robots started the mission as one SoNS and, after failure occurred, the SoNS continued the mission with the robots still available. The full set of results shows that when ground robots failed, the rest of the robots were able to stay connected in one SoNS and complete the mission, whereas when an aerial robot failed, some of the ground robots downstream from it were disconnected from the primary SoNS, but the remaining connected robots were able to continue the mission. For example, in the trial shown in Fig. 4.17A,B, one of the aerial robots failed (see purple star in A) at approx. 85 s (see red dotted line in B). The only aerial robot that remained functional recruited the ground robots that remained

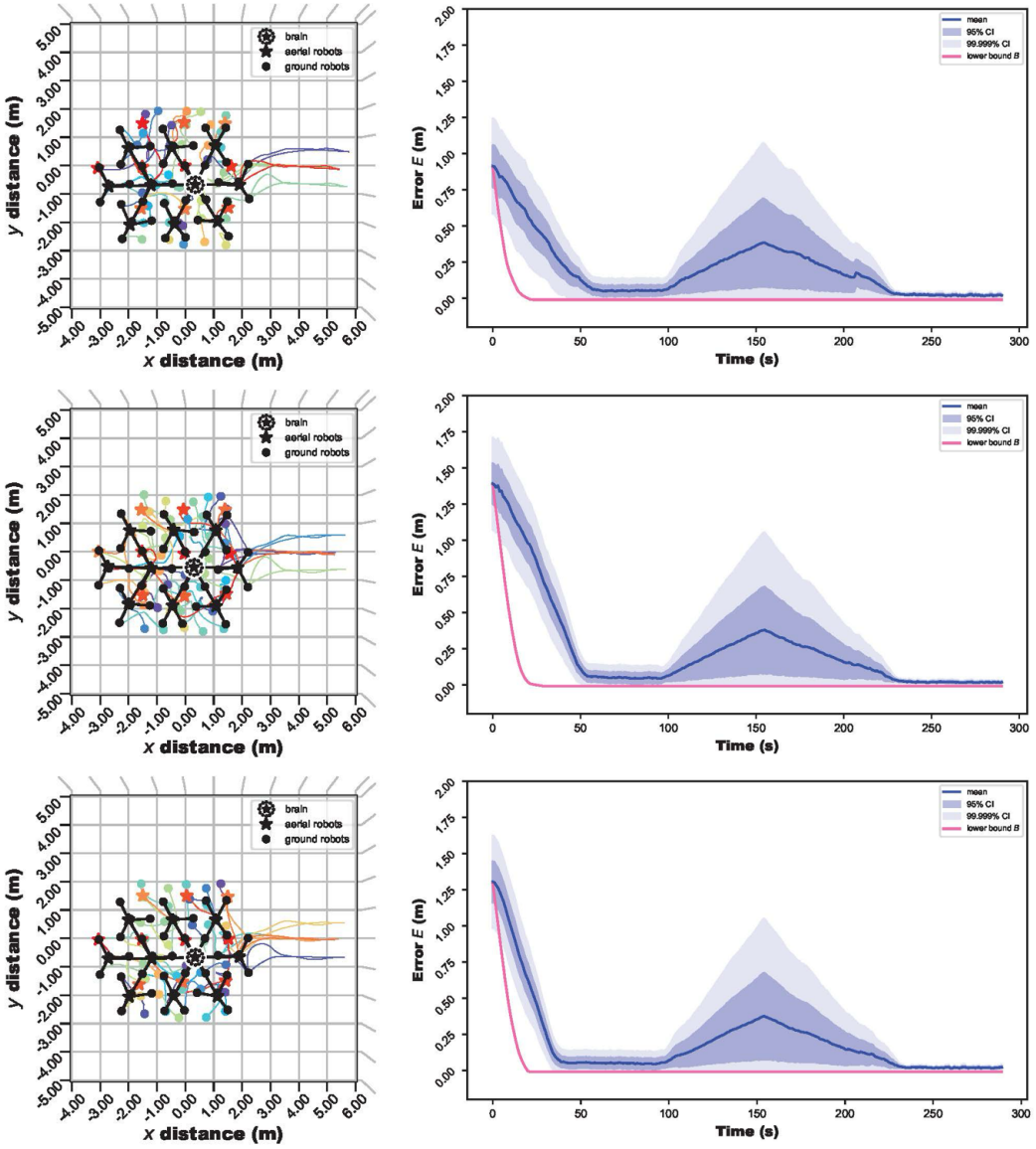


Figure 4.16: Splitting and merging systems, simple split and merge: Example simulation trials. This variant consists simply of split and merge operations, without search-and-rescue or other constituent tasks. 50 trials were conducted in simulation, each with 50 robots. (Reprinted from (Zhu et al., 2024).)

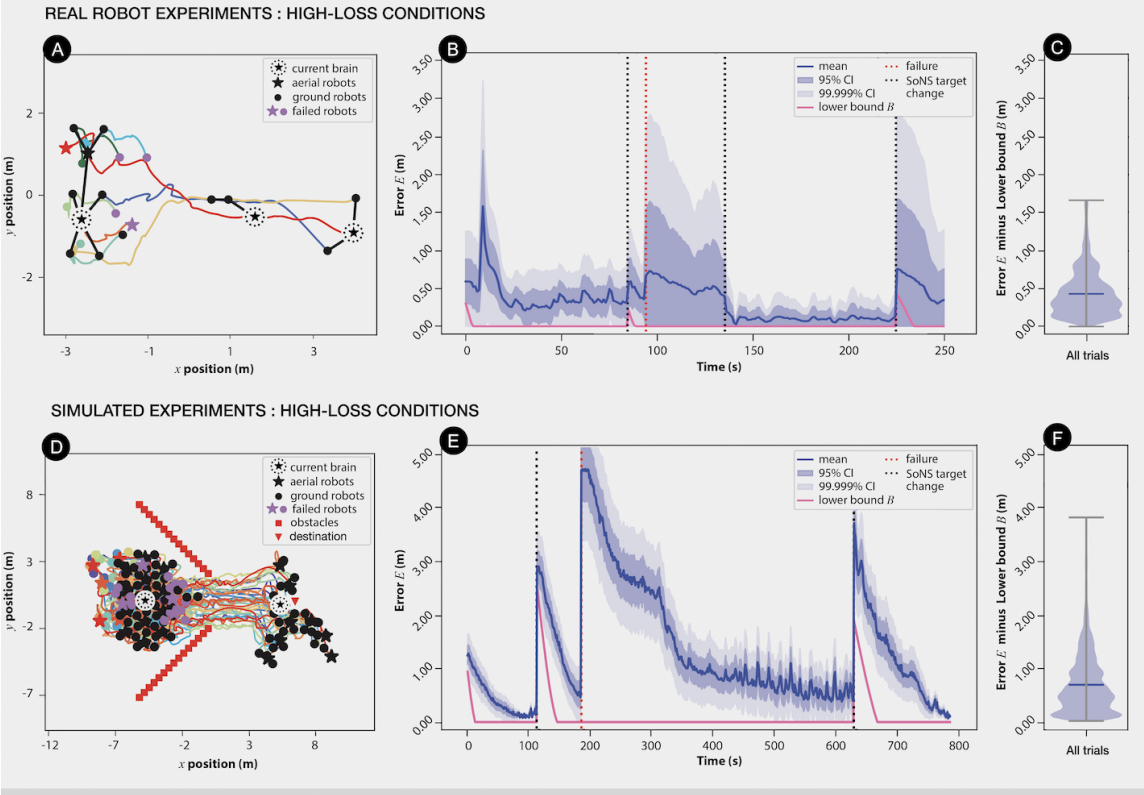


Figure 4.17: Fault tolerance. Results with real robots and in simulation, testing both permanent failures (up to two-thirds of robots) and temporary system-wide failures of communication or vision. (A-C) High-loss conditions with real robots: arbitrary permanent failures of multiple robots in an 8-robot SoNS. **(B,C)** In a real robot example trial: (A) trajectories of robots over time, with the initial (left), an example intermediate, and final (right) SoNS indicated in black, with the failed robots indicated in purple at their shutdown positions, and (B) mean actuation error E (Eq. 4.40) with lower bound B (Eq. 4.42) plotted for reference. (C) Violin plot of E (Eq. 4.40) minus B (Eq. 4.42) in all five trials. (D-F) High-loss conditions in simulation: $33.\overline{3}\%$ or $66.\overline{6}\%$ failure probabilities in a 65-robot SoNS. **(B,C)** In a simulated example trial with $66.\overline{6}\%$ probability for each robot to fail: (D) trajectories of robots over time, with the initial (left) and final (right) SoNS indicated in black, with the failed robots indicated in purple at their shutdown positions, and (E) mean actuation error E (Eq. 4.40) with lower bound B (Eq. 4.42). (F) Violin plot of $E-B$ in all trials with both $33.\overline{3}\%$ and $66.\overline{6}\%$ failure probabilities, 50 trials per probability. (Reprinted from (Zhu et al., 2024).)

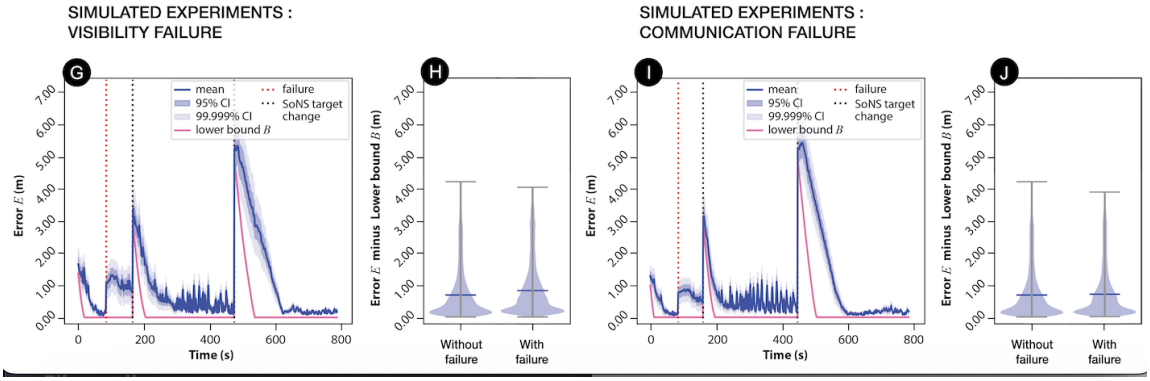


Figure 4.18: Fault tolerance. Results with real robots and in simulation, testing both permanent failures (up to two-thirds of robots) and temporary system-wide failures of communication or vision. (G-H) System-wide vision failure for 30 s in simulation, in a 65-robot SoNS: (G) E in an example trial and (H) violin plots of $E - B$ in all 50 trials, compared to 50 trials without failure. (I-J) System-wide communication failure for 30 s in simulation, in a 65-robot SoNS: (I) E in an example trial and (J) violin plots of $E - B$ in all 50 trials, compared to 50 trials without failure. (Reprinted from (Zhu et al., 2024).)

functional and reachable, and continued with the mission, eventually reaching the object marking the final destination. In all trials in which at least one robot of each type remained functional, the SoNS was able to reorganize itself with the remaining robots and continue with the mission, resulting in relatively low overall error for all trials (Fig. 4.17C).

In simulation, we ran the same setup with a larger swarm of 65 robots (Fig. 4.17D-F), to test task performance under high-loss conditions. In these experiments, each robot had probability p to fail, regardless of hardware type or brain status. We tested two variants by setting $p = 0.\overline{3}$ or $p = 0.\overline{6}$, with 50 trials per variant. In the example trial in Fig. 4.17D-F, after two-thirds of robots failed, including the brain, the SoNS continued with the mission and eventually reached the final destination object. In all trials, the SoNS was able to reorganize itself with available robots (i.e., robots that had not failed and were not stuck in place), returning to a low steady-state error (Fig. 4.17E).

Also in simulation in a swarm of 65 robots, we tested two types of temporary system-wide failures that would be likely to occur in practice (Fig. 4.18G-J): vision failure (for example, because of an obstruction in the environment) and wireless communication failure. We tested visibility and wireless communication failures with durations of 0.5 s, 1.0 s, and 30 s, with 50 trials per duration for each failure type. In all trials, the SoNS was able to re-establish itself after the system-wide failure, continuing with the mission and leaving behind less than 5% of robots (i.e., three robots or less) in any trial. In all trials, the actuation error increased after failure occurred (see red dotted lines in Fig. 4.18G,I) but by the end of the mission returned to low steady-state error.

The fault tolerance setups include four variants (see Figs. 4.19-4.23) that are all based on the same robot mission, with one variant run in experiments with real robots and the other three run in simulation.

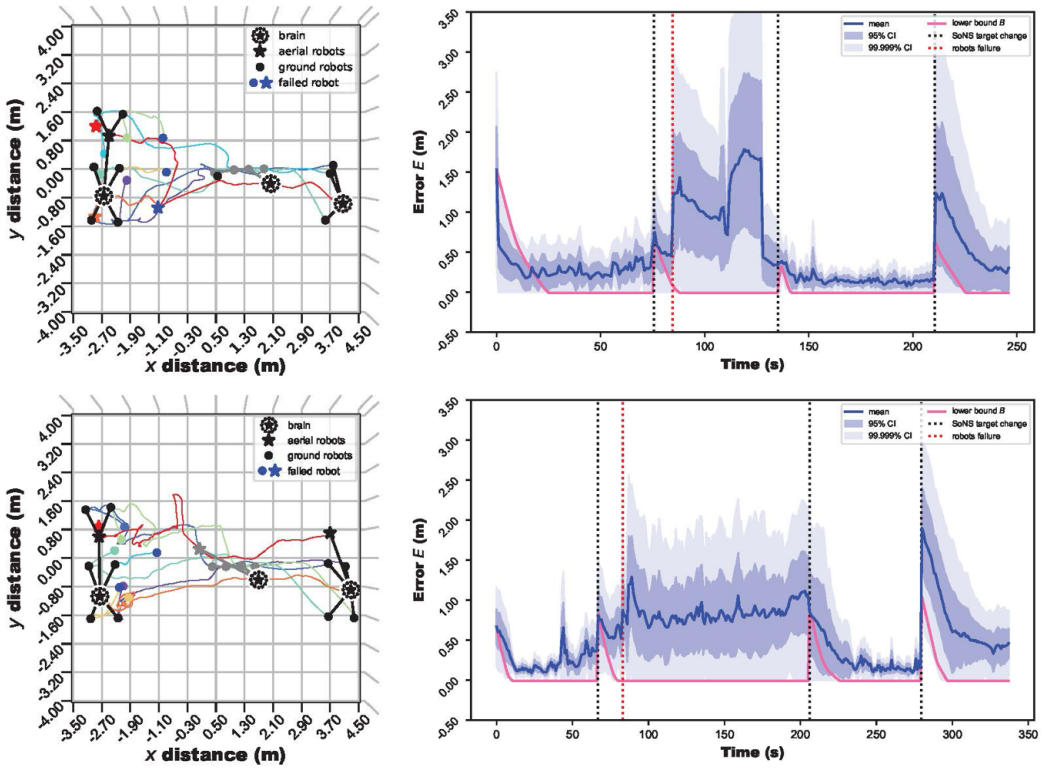


Figure 4.19: Multiple permanent failures: Real robot trials. Five trials with real robots were conducted, each with eight robots. Failures are triggered uniformly randomly. Note that in the third trial shown here, both aerial robots failed, and therefore the remaining functional robots (all ground robots) did not fulfill the requirements of the mission and did not continue with the mission after the failures occurred *(figure continued on next page)*. (Reprinted from (Zhu et al., 2024).)

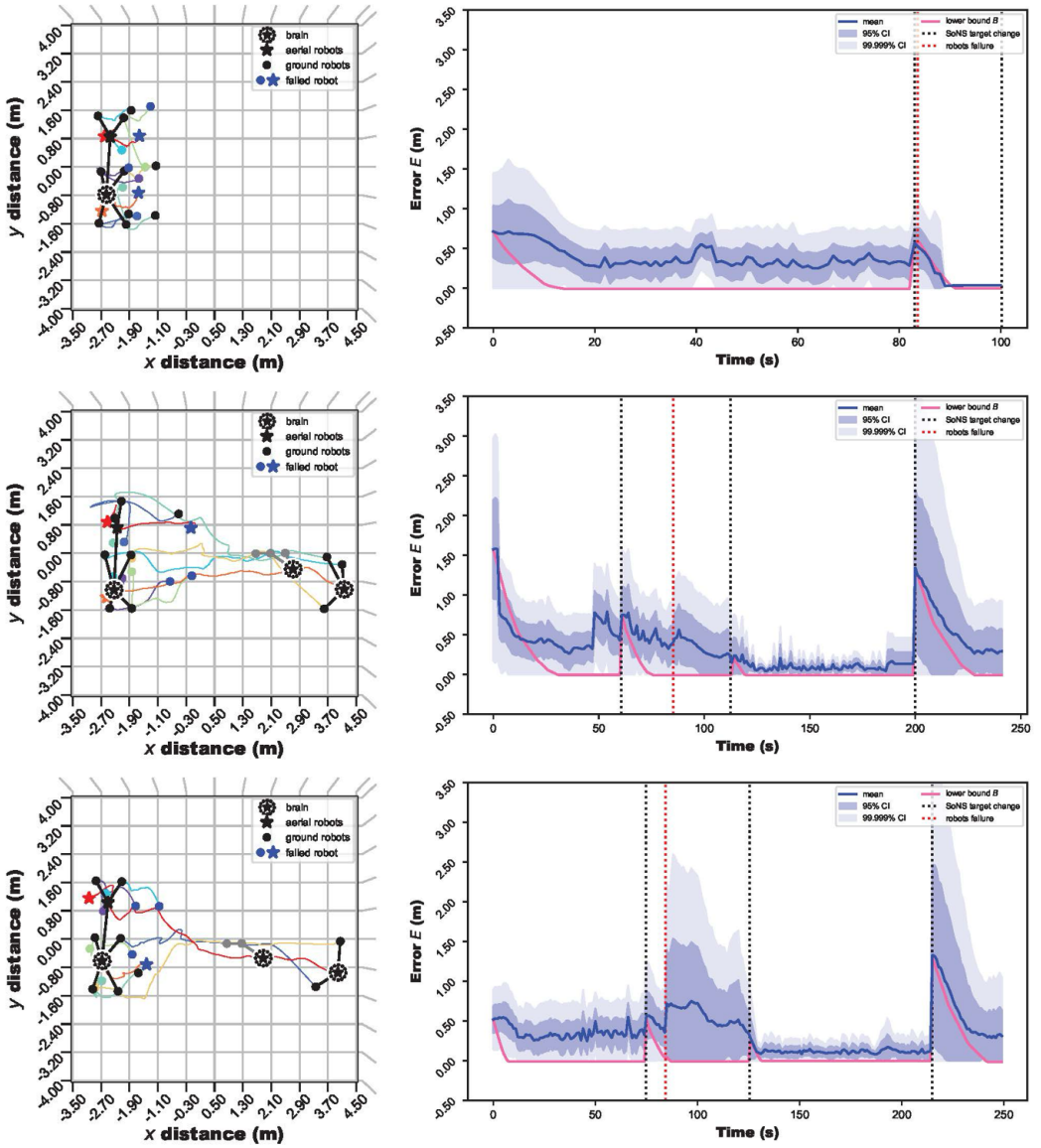


Figure 4.19: *(cont'd)* Multiple permanent failures: Real robot trials. Five trials with real robots were conducted, each with eight robots. Failures are triggered uniformly randomly. Note that in the third trial shown here, both aerial robots failed, and therefore the remaining functional robots (all ground robots) did not fulfill the requirements of the mission and did not continue with the mission after the failures occurred. (Reprinted from (Zhu et al., 2024).)

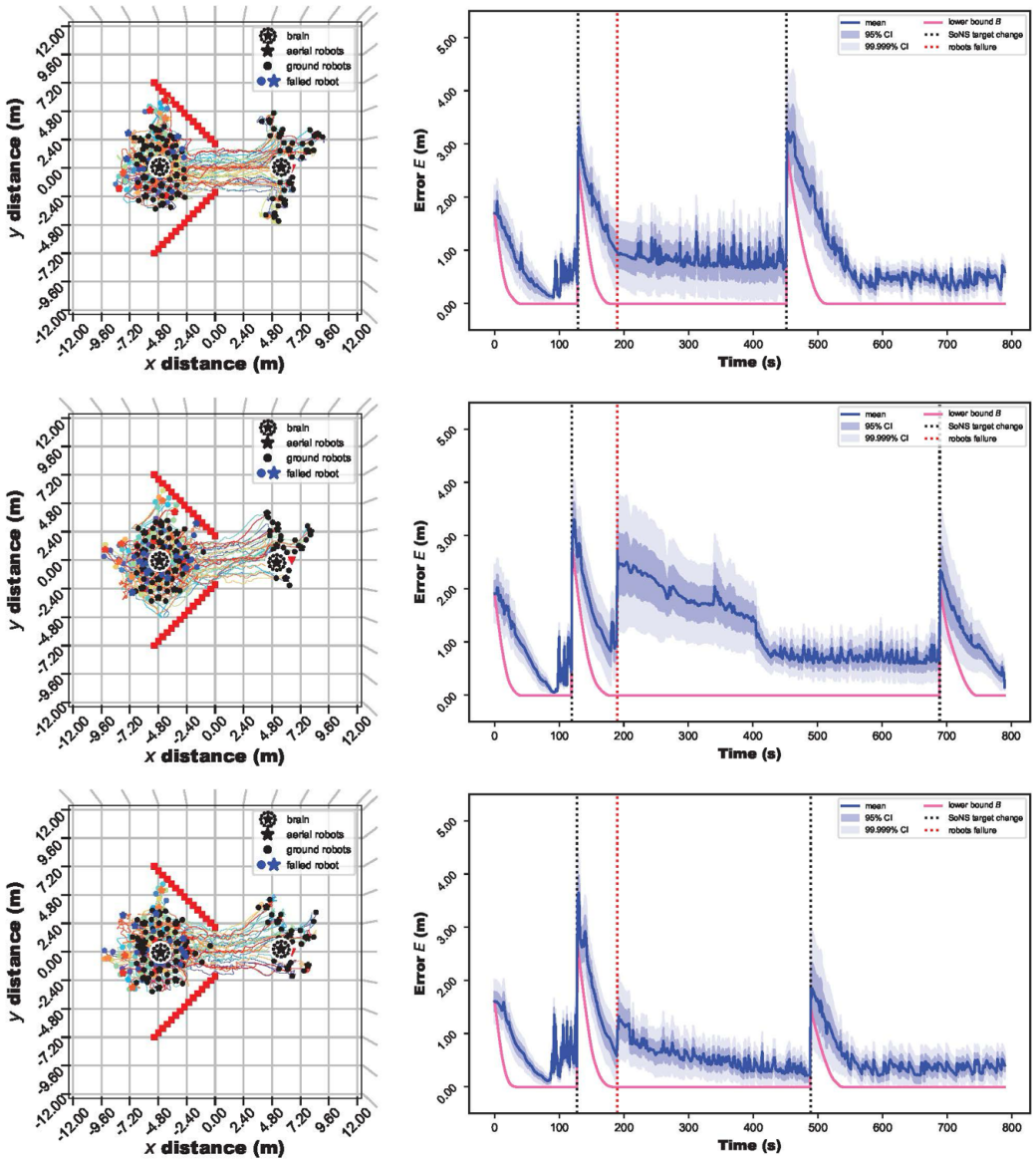


Figure 4.20: High-loss conditions, $33.\overline{3}\%$ probability to fail: Example simulation trials. 50 trials were conducted in simulation, each with 65 robots. (Reprinted from (Zhu et al., 2024).)

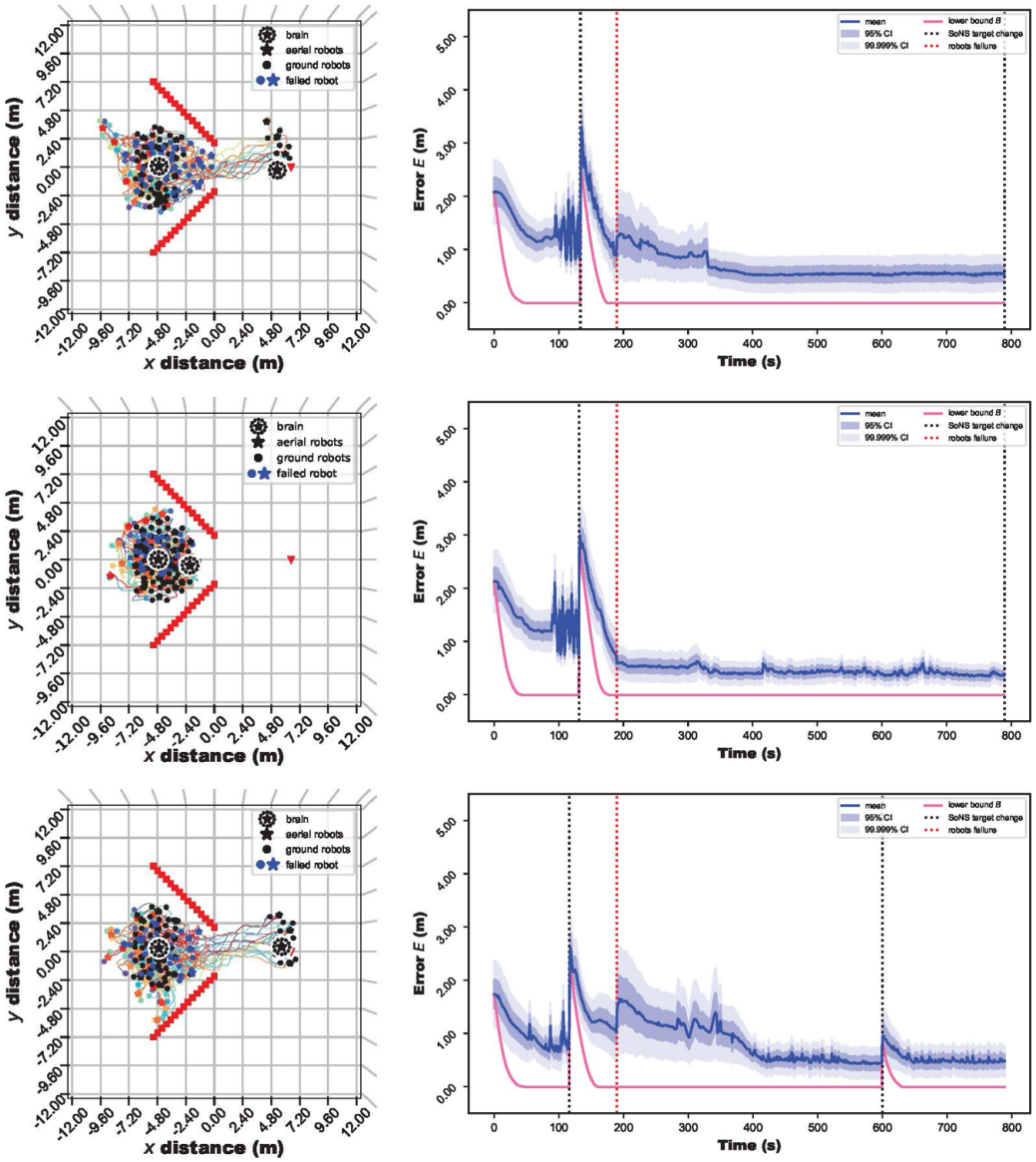


Figure 4.21: High-loss conditions, $66.\overline{6}\%$ probability to fail: Example simulation trials. Note that in some trials under this condition, the SoNS was able to re-establish itself after the failures occurred, but was not able to complete other portions of the mission due to, for example, functional ground robots in the SoNS becoming trapped by the failed ground robots (as in the middle trial shown here). 50 trials were conducted in simulation, each with 65 robots. (Reprinted from (Zhu et al., 2024).)

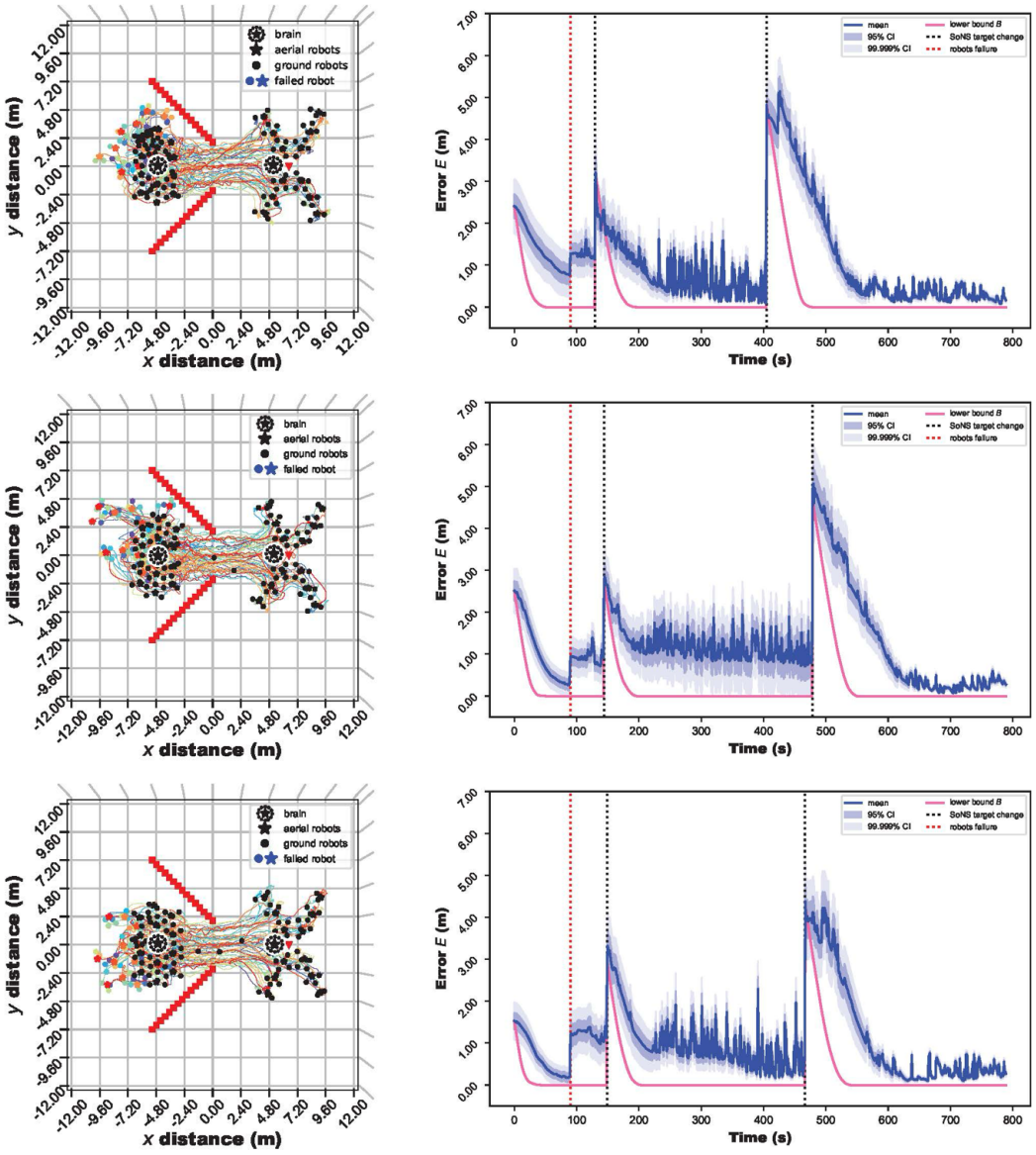


Figure 4.22: Temporary system-wide vision failure: Example simulation trials of three different failure durations (from top to bottom, 0.5, 1.0, and 30 s). 50 trials per duration were conducted in simulation, each with 65 robots. (Reprinted from (Zhu et al., 2024).)

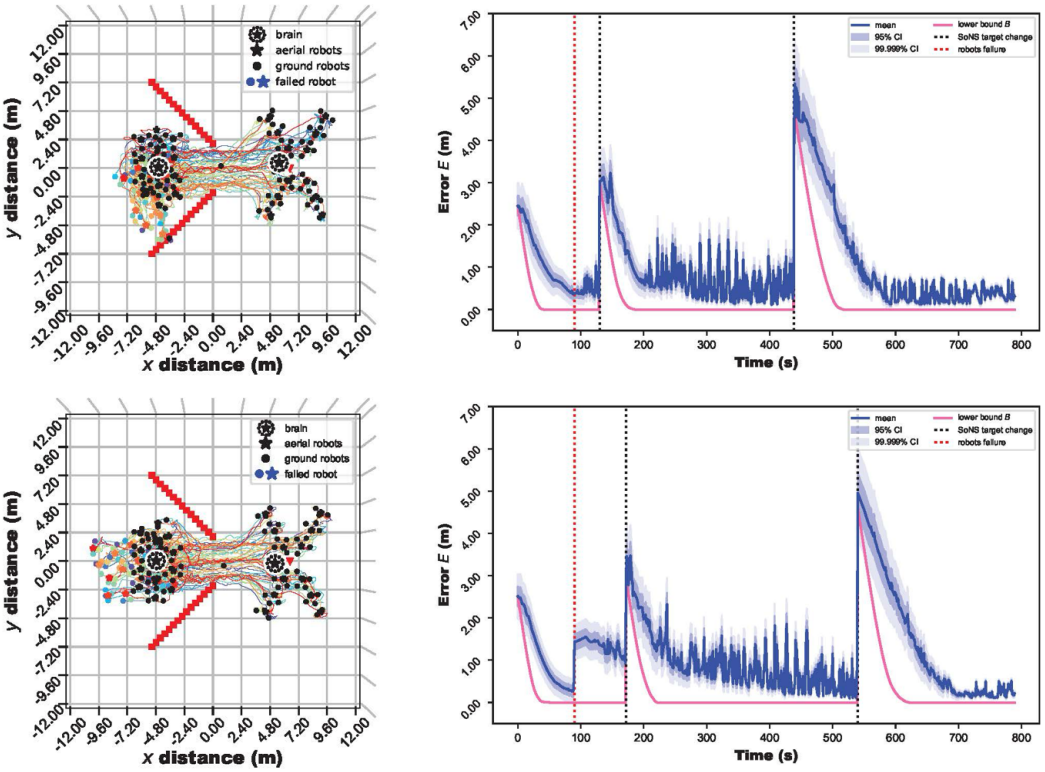


Figure 4.23: Temporary system-wide communication failure: Example simulation trials of three different failure durations (from top to bottom, 0.5, 1.0) 50 trials per duration were conducted in simulation, each with 65 robots. (Reprinted from (Zhu et al., 2024).)

4.5 Chapter conclusions

The results demonstrate that the SoNS approach greatly expands the state of the art in swarm robotics, enabling qualitatively new swarm behaviors and behavior combinations. Using the SoNS approach, we have shown that robot swarms can self-reconfigure their communication, control, and behavior structures on demand, coordinating sensing, actuation, and decision making SoNS-wide without sacrificing agility, scalability, or fault tolerance. Finally, the proof-of-concept missions reflect the relative ease of behavior design provided by the SoNS approach, allowing a user to program a whole swarm as if it were a single robot with a reconfigurable morphology.

Firstly, the self-organized controllable hierarchy of the SoNS approach, which is novel for robot swarms, is demonstrated in all missions through the maintenance and reconfiguration of hierarchical communication structures. The topology and relative positions are both controllable (sometimes separately), without any external intervention. The second novel feature of the SoNS approach—its self-organization of explicit and interchangeable leadership roles—enables robot systems to establish and reorganize themselves flexibly, without having to reinitialize the whole architecture if, for example, a brain loses connection or some of the robots at an inner hierarchy level experience environmental disturbances. This feature can be seen in the fault tolerance experiments, in which SoNSs are able to automatically reorganize after a brain is lost or a majority of robots fail, continuing their mission without having to backtrack on past progress or synchronize with an external reference. Thirdly, inter-system reconfiguration is demonstrated during all SoNS establishment operations and splitting and merging operations, such as in the splitting and merging mission. The fourth novel feature of the SoNS approach is the ability to reconfigure the system's internal behavior structures, for instance by renegotiating the inter-level control distribution.

In short, we have shown the SoNS system architecture to provide a reconfigurable explicit control hierarchy for robot swarms, which is in itself novel. Beyond that, because the SoNS approach allows the system architecture to be multi-level and dynamic while still being self-organized, it enables qualitatively new behaviors and

behavior combinations in robot swarms. Using the SoNS approach, we have shown that robot swarms can handle increased mission complexity, self-reconfiguring their communication, control, and behavior structures on demand. We have also shown that the SoNS approach retains the fault tolerance advantage for which robot swarms are often studied. Any robot, including the brain, is shown to be immediately and automatically replaceable in both reality and simulation, and SoNSs are able to recover and continue missions in high-loss conditions, as well as recover from temporary system-wide failures of detection or communication. In these fault tolerance experiments, we specifically tested a SoNS's ability to recover without adapting its overall mission, to assess baseline performance. If robots had instead been allowed to search for each other after experiencing disconnections (as shown in the search-and-rescue mission), then more robots could have been retained. Furthermore, the communication plateaus at less than 0.6 kB per robot, which in SoNSs of up to 250 robots is less than 120 kB of messages per step, remaining well below the bandwidth limitations of several types of wireless communication technologies. Overall, the results show that, using the SoNS, sensing, actuation, and decision making can be coordinated SoNS-wide, without sacrificing scalability, flexibility, and fault tolerance.

Chapter 5

Reconfigurable Frameworks for Self-organized Hierarchy

The previous chapter introduced a self-organizing hierarchical swarm architecture and conducted comprehensive experiments for its validation. This chapter[1] *builds on the theoretical foundations required to use this architecture for the formation control of large swarms. We focus on the systematic and mathematically analyzable organization and reorganization of hierarchical frameworks in a robot swarm. Although methods for framework construction and formation maintenance via rigidity theory exist in the literature, they do not address cases of hierarchy in a robot swarm. To achieve this, in this chapter, we extend bearing rigidity to directed topologies and expand Henneberg constructions to generate self-organized hierarchical topologies with bearing rigidity. We investigate three key self-reconfiguration problems: framework merging, robot departure, and framework splitting. We derive the mathematical conditions for these problems and develop algorithms that preserve rigidity and hierarchy using only local information. Our approach can be used for formation control generally, as in principle it can be coupled with any control law that makes use of bearing rigidity. To demonstrate and validate our proposed hierarchical frameworks and methods, we apply them to four scenarios of reactive formation control using an example control law.*

[1]The content of this chapter was previously published in (Zhang et al., 2023).

5.1 Approach and contributions

This chapter addresses self-reconfigurable hierarchical frameworks in robot swarms, for the purpose of formation control, enabled by bearing rigidity theory and the notion of hierarchy. Our hierarchical frameworks can construct and reconstruct themselves comprehensively (through robot addition, framework merging, robot departure, and framework splitting) using only local measurement and local communication. To demonstrate our hierarchical frameworks and validate our theoretical results, we implemented the frameworks within a simple controller and evaluated it across four experimental scenarios.

The main technical contributions of this chapter can be summarized as follows:

1. We extend bearing rigidity theory (Zhao and Zelazo, 2019) to directed graphs with lower triangular structure. This is non-trivial because of the analysis difficulty associated with the lack of symmetry, which must be handled in directed graphs. We also show that to evaluate rigidity under such a topology, in addition to the concept of *infinitesimal bearing rigidity*, the concept of *bearing persistence* is required. We provide the necessary and sufficient conditions to uniquely determine a framework under a directed topology with asymmetric and lower triangular structure.

2. We propose a novel Hierarchical Henneberg Construction (HHC) to integrate the concepts of bearing rigidity and hierarchy. Compared to the bearing-based Henneberg Construction proposed in (Trinh et al., 2018), which analyzes rigidity from a geometric perspective using a global reference frame, our method can establish rigidity intrinsically and in a decentralized way (i.e, without global references). The only existing approach to analyze bearing rigidity in a decentralized way without a global reference is (Schiano et al., 2016), which has few constraints on topology, but still, assumes that the topology has already been established. Our approach, by contrast, uses hierarchy to construct and maintain the necessary topology and intrinsically establish rigidity, without relying on an absolute reference, system-wide broadcast, or other global mechanisms.

3. We propose the mathematical conditions and design the distributed algorithms to preserve rigidity and hierarchy during framework construction (including robot addition), framework merging, robot departure, and framework splitting.

5.2 From bearing rigidity to bearing persistence

Notation: $\mathbb{R}^d$ is the d-dimensional Euclidean space. $\mathbf{0}$ is a zero matrix with appropriate dimension; The $d \times d$ identity matrix is denoted by $\mathbf{I}_d$ and the $n \times 1$ vector of all ones is denoted by $\mathbf{1}_n$. $\operatorname{rank}(\cdot)$ and $\operatorname{Null}(\cdot)$ are the rank and null space of a square matrix; $\operatorname{card}(\cdot)$ denotes the number of elements in a set. $\|\cdot\|$ is the Euclidean norm of a vector.

Consider a set of n ($n \geq 2$) robots in $\mathbb{R}^d$ ($d = 2, 3$). $p_i(t) \in \mathbb{R}^d$ denotes the position of robot $i \in \{1, 2, \ldots, n\}$ at time t and the vector $p(t) = \left[p_1^T(t), p_2^T(t), \ldots, p_n^T(t)\right]^T \in \mathbb{R}^{dn}$ describes the configuration of the robot swarm at time t. Interactions among the robots are characterized by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $|\mathcal{V}| = n$ and $|\mathcal{E}| = m$. If $e_{ji} = (e_j, e_i) \in \mathcal{E}$, then the i-th robot can receive information from the j-th robot. $\mathcal{G}$ is undirected if $\forall e_{ji} \in \mathcal{E}$, there exists $e_{ij} \in \mathcal{E}$; otherwise, $\mathcal{G}$ is directed. We define the parent set of vertex v_i as $\mathcal{P}_i = \{v_j \in \mathcal{V} | e_{ji} \in \mathcal{E}\}$, and the child set of v_i as $\mathcal{C}_i = \{v_j \in \mathcal{V} | e_{ij} \in \mathcal{E}\}$.

It is assumed that for each edge $e_{ji} \in \mathcal{E}$, robot i can continuously measure the bearing of robot j where the bearing vector is $g_{ij} = p_{ij}/\|p_{ij}\|$ and where $p_{ij} = p_j - p_i$ is the displacement vector.

We define a *framework* as a graph $\mathcal{G}$ associated with a configuration p, i.e., $(\mathcal{G}, p)$. According to whether the underlying graph is directed or not, the framework is either an *undirected framework* or a *directed framework*.

In the next subsection we will recall some classical concept concerning the so-called bearing rigidity of undirected frameworks, and in the subsequent subsection we will report a series of new results on directed frameworks that will be used in this chapter.

5.2.1 Bearing rigidity in undirected frameworks

To describe all the bearings in $(\mathcal{G}, p)$, define the bearing function F_B as $F_B(p) \triangleq \left[g_1^T, g_2^T, \ldots, g_m^T\right]^T$, where g_k corresponds to the k-th edge in graph $\mathcal{G}$. Then, we can define the *bearing rigidity matrix* as

$$R_B(p) \triangleq \frac{\partial F_B}{\partial p} \in \mathbb{R}^{dm \times dn} \tag{5.1}$$

Definition 1 – Infinitesimal bearing rigidity (Zhao and Zelazo, 2019): An undirected framework $(\mathcal{G}, p)$ in $\mathbb{R}^d$ is infinitesimally bearing rigid (IBR) if and only if the positions of all robots in the framework can be uniquely determined up to a translational and scaling factor.

Throughout this chapter, we refer to frameworks that satisfy Definition 1 as infinitesimally bearing rigid (IBR), and to those that do not as non-infinitesimally bearing rigid (non-IBR).

Lemma 1 (Zhao and Zelazo, 2019): An undirected framework $(\mathcal{G}, p)$ in $\mathbb{R}^d$ is IBR if and only if $\operatorname{rank}(R_B(p)) = dn - d - 1$, or equivalently $\operatorname{Null}(R_B(p)) = \operatorname{span}\{\mathbf{1}_n \otimes \mathbf{I}_d, p\}$.

Another equivalent definition for an IBR framework is that all the infinitesimal bearing motions are trivial[2], i.e., translation and scaling are the only robot motions that preserve the relative bearings between robots connected by an edge. Examples of *non-infinitesimally bearing rigid frameworks* are presented in Fig. 5.1(a), where there clearly exist non-trivial infinitesimal bearing motions (see red dashed arrows), under which the framework will deform. By contrast, Fig. 5.1(b) shows examples of rigid frameworks where the only infinitesimal motions possible are rigid translation and scaling of the frameworks. Note that the cases reported in Fig. 5.1(b) are obtained by rigidifying the examples in Fig. 5.1(a), by adding edges (see blue edges) to eliminate non-trivial infinitesimal bearing motions.

[2]Two kinds of trivial infinitesimal bearing motions exist: translational and scaling of the entire framework. More details are given in (Zhao and Zelazo, 2019).

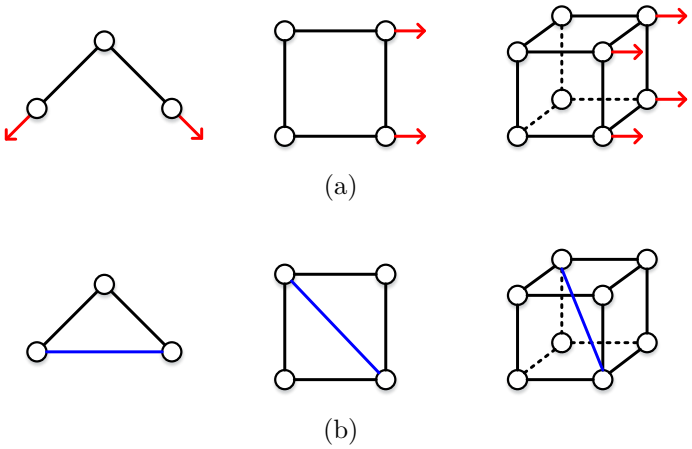


Figure 5.1: (a) Examples of *non-IBR* frameworks. The red arrows represent non-trivial infinitesimal bearing motions, under which the framework will deform and cannot be uniquely determined. (b) Examples of *IBR* frameworks. In contrast to examples in (a), the newly added edges (blue edges) can eliminate non-trivial infinitesimal bearing motions, such that the configuration is uniquely determined. (Reprinted from (Zhang et al., 2023)).

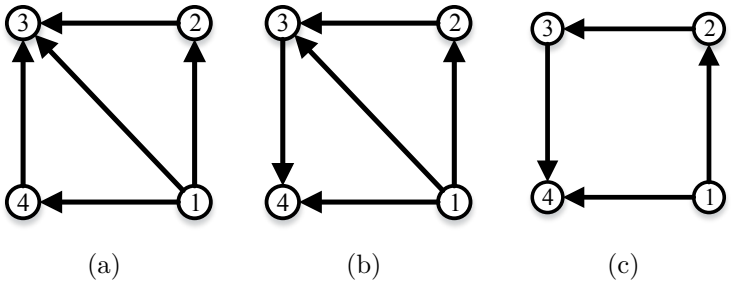


Figure 5.2: Examples to illustrate the concepts of bearing rigidity (Zhao and Zelazo, 2015) and bearing persistence in directed frameworks. (a) An infinitesimally bearing rigid (IBR) but non-bearing persistent (non-BP) framework. (b) A framework that is both IBR and bearing persistent (BP). (c) A *Non-IBR* but *BP* framework. (Reprinted from (Zhang et al., 2023).)

5.2.2 Bearing persistence in directed frameworks

It is worth noting that rigidity is fundamentally an *undirected* notion, and therefore is not sufficient to characterize *directed* frameworks (Hendrickx et al., 2007). Consider the framework in Fig. 5.2(a). Although it is IBR (because of the rank of R_B), this framework cannot always be determined uniquely. In this framework, Robot 1 has no bearing constraints, therefore it can be placed arbitrarily in space. After the position of Robot 1 is determined, Robot 2 and Robot 4 can be subsequently placed. However, Robot 2 and Robot 4 only have one bearing constraint and they can be randomly placed along edges e_{12} and e_{14}. Once the positions of Robots 1, Robot 2, and Robot 4 are determined, it is clear that the position of Robot 3 is not always feasible, because it has three bearing constraints to be satisfied. The position of Robot 3 is feasible if and only if $\|p_{21}\| = \|p_{41}\|$. Therefore, rigidity is not sufficient to characterize the framework in Fig. 5.2(a). By contrast, the framework in Fig. 5.2(b) can be uniquely determined as an undirected framework.

This example indicates that more conditions are required to guarantee the existence and uniqueness of a directed framework. Therefore, in this chapter we will also use the condition of *bearing persistence* (BP). Before defining this notion, we introduce another bearing-related matrix $B \in \mathbb{R}^{dn\times dn}$, namely the *bearing Laplacian*, which is defined as (Zhao and Zelazo, 2019)

$$B_{ij} = \begin{cases} \mathbf{0}, & i \neq j, e_{ji} \notin \mathcal{E} \\ -P_{g_{ij}}, & i \neq j, e_{ji} \in \mathcal{E} \\ \sum_{v_k \in \mathcal{P}_i} P_{g_{ik}}, & i = j \end{cases} \tag{5.2}$$

where $B_{ij} \in \mathbb{R}^{d\times d}$ is the ijth block of a submatrix of B, and $P_{g_{ij}}$ is an orthogonal projection operator defined as $P_{g_{ij}} \triangleq \mathbf{I}_d - g_{ij}g_{ij}^T$. It can be proved that $P_{g_{ij}}$ is positive semi-definite, 0 is a simple eigenvalue of $P_{g_{ij}}$, $\text{Null}\left(P_{g_{ij}}\right) = \text{span}\,(p_i - p_j)$, and $\text{rank}\left(P_{g_{ij}}\right) = d - 1$.

Lemma 2 (Zhao et al., 2017): $\text{rank}\,(B_{ii}) = d$ if and only if there exist at least two vertices $v_j, v_k \in \mathcal{P}_i$ such that $g_{ij} \neq g_{ik}$. (Zhao and Zelazo, 2015): A directed

framework $(\mathcal{G}, p)$ in $\mathbb{R}^d$ is bearing persistent (BP) if $\text{Null}\,(B) = \text{Null}\,(R_B)$. Frameworks that do not satisfy this property will be referred to as non-bearing persistent (non-BP).

For *undirected* frameworks, the bearing Laplacian matrix B is symmetric positive semi-definite, which satisfies $\text{Null}\,(B) = \text{Null}\,(R_B)$ (Zhao and Zelazo, 2015). For *directed* frameworks, however, only $\text{Null}\,(R_B) \subset \text{Null}\,(B)$ is guaranteed. Note that bearing persistence is independent of rigidity. An example is illustrated in Fig. 5.2(c), which is not IBR but is still BP.

Even when using persistence, whether all IBR and BP directed frameworks can be uniquely determined is still an open problem (Zhao and Zelazo, 2019). This research focuses on directed graphs with a hierarchical structure. Specifically, each robot only observes and tracks two immediate neighbors with higher hierarchy (i.e., parents), which results in a lower triangular structure (cf. (Nagy et al., 2010)). Therefore, the bearing Laplacian of these special directed graphs can be written as

$$B = \begin{bmatrix} \mathbf{0} & & & \mathbf{0} \\ B_{2,1} & B_{2,2} & & \\ \vdots & \vdots & \ddots & \\ B_{n,1} & B_{n,2} & \cdots & B_{n,n} \end{bmatrix} \tag{5.3}$$

Lemma 3: Consider a directed framework $(\mathcal{G}, p)$ in $\mathbb{R}^d$. If the corresponding bearing Laplacian matrix B is lower triangular, the following statements are equivalent.

(1) $(\mathcal{G}, p)$ is IBR and BP.

(2) $(\mathcal{G}, p)$ can be uniquely determined up to a translational and scaling factor.

(3) $\text{Null}\,(B) = \text{Null}\,(R_B) = \text{span}\,\{\mathbf{1}_n \otimes \mathbf{I}_d, p\}$.

(4) $\text{rank}\,(B) = dn - d - 1$.

(5) $\text{rank}\,(B_{2,2}) = d - 1$ and $\text{rank}\,(B_{ii}) = d, \forall i \geq 3$.

Proof: According to the definitions of infinitesimal bearing rigidity and bearing persistence, (1) $\Leftrightarrow$ (3) $\Leftrightarrow$ (4) is straightforward. We therefore only show (2) $\Leftrightarrow$ (3) and (4) $\Leftrightarrow$ (5).

(2) $\Rightarrow$ (3): To demonstrate $\text{Null}(B) = \text{span}\{\mathbf{1}_n \otimes \mathbf{I}_d, p\}$, it is equivalent to show $\forall q = \left[q_1^T, \cdots, q_n^T\right]^T \in \text{Null}(B)$, $q = ap + \mathbf{1}_n \otimes b$, where $a \in \mathbb{R} \setminus \{0\}$ and $b \in \mathbb{R}^d$.

For robot 1, q_1 can be chosen randomly according to Eq. (5.3), thus it is always possible to find a and b, such that $q_1 = ap_1 + b$.

For robot 2, q_2 satisfies $B_{2,1}(q_2 - q_1) = \mathbf{0}$. If $B_{2,1} = \mathbf{0}$, there is no bearing constraint for robot 2, thus robot 2 can be placed randomly, which contradicts statement (2). Therefore, $q_2 - q_1 \in \text{Null}(B_{2,1}) = \text{span}\{p_2 - p_1\}$, i.e., $q_2 - q_1 = \alpha(p_2 - p_1)$ with $\alpha \in \mathbb{R} \setminus \{0\}$. Now, we claim that $q_i = q_1 + \alpha(p_i - p_1)$ for all $1 \leq i \leq n$, and use mathematical induction to check whether this claim is true.

For robot 3, the constraint is $(B_{3,1} + B_{3,2})\, q_3 = B_{3,1}q_1 + B_{3,2}q_2$. Using $q_2 - q_1 = \alpha(p_2 - p_1)$, the constraint can be rewritten as

$$\begin{aligned}(B_{3,1} + B_{3,2})\, q_3 &= (B_{3,1} + B_{3,2})\, q_1 + \alpha B_{3,2}(p_2 - p_1) \\ &= (B_{3,1} + B_{3,2})\,(q_1 + \alpha(p_3 - p_1))\end{aligned} \tag{5.4}$$

where the last equality uses $B_{3,1}(p_3 - p_1) + B_{3,2}(p_3 - q_2) = 0$. Under Lemma 2, $B_{3,1} + B_{3,2}$ is not singular if and only if $g_{3,1}$ and $g_{3,2}$ exist, and are not collinear. If $g_{3,1}$ (or $g_{3,2}$) does not exist, robot 3 only has one bearing constraint, thus it has a non-infinitesimal bearing motion (such as robot 3 in Fig. 5.2(c)), thus the framework cannot be uniquely determined. If $g_{3,1}$ are collinear with $g_{3,2}$, robot 3 still only has one bearing constraint, and the framework will not be unique. This implies that $B_{3,1} + B_{3,2}$ is not singular, and we obtain $q_3 = q_1 + \alpha(p_3 - p_1)$.

Now, we assume that $q_k = q_1 + \alpha(p_k - p_1)$ is true for $1 \leq k \leq i-1$. For robot i, we have

$$\begin{aligned}\textstyle\sum_{j=1}^{i-1} B_{ij} q_i &= \textstyle\sum_{j=1}^{i-1} B_{ij} q_j \\ &= \textstyle\sum_{j=1}^{i-1} B_{ij} q_1 + \alpha \sum_{j=1}^{i-1} B_{ij}(p_j - p_1) \\ &= \textstyle\sum_{j=1}^{i-1} B_{ij}(q_1 + \alpha(p_i - p_1))\end{aligned} \tag{5.5}$$

where the last equality uses $\sum_{j=1}^{i-1} B_{ij}(p_i - p_j) = 0$. Via similar analysis for robot 3, the uniqueness of the framework ensures that $\sum_{j=1}^{i} B_{ij}$ is non singular, and further that $q_i = q_1 + \alpha(p_i - p_1)$.

By the above induction, we prove that $q_i = q_1 + \alpha(p_i - p_1)$ is true for all robots. Moreover, it can be derived that $q = \alpha p + \mathbf{1}_n \otimes (q_1 - \alpha p_1)$. This implies that $\forall q \in \mathrm{Null}(B)$, $q \in \mathrm{span}\{\mathbf{1}_n \otimes \mathbf{I}_d, p\}$.

(3) $\Rightarrow$ (2): Consider an IBR and BP framework $(\mathcal{G}, p)$. For a configuration $q \in \mathbb{R}^n$, we say q is a realization of directed graph $\mathcal{G}$, if $P_{p_i - p_j}(q_i - q_j) = 0$ for all $e_{ij} \in \mathcal{E}$. Denote the set of all realizations of $\mathcal{G}$ as $S_{\mathcal{G}}$. Our objective is to demonstrate that $\forall q \in S_{\mathcal{G}}$, $q \in \mathrm{span}\{\mathbf{1}_n \otimes \mathbf{I}_d, p\}$. This can be directly verified via the bearing Laplacian. Given that

$$Bq = \begin{bmatrix} \vdots \\ \sum_{j=1}^{i-1} B_{ij}(q_i - q_j) \\ \vdots \end{bmatrix} = \mathbf{0} \tag{5.6}$$

then $q \in \mathrm{Null}(\mathrm{B}) = \mathrm{span}\{\mathbf{1}_n \otimes \mathbf{I}_d, p\}$.

(4) $\Rightarrow$ (5): According to Eq. (5.2), $\mathrm{rank}(B_{2,2}) = d-1$ if $e_{12} \in \mathcal{E}$, and $\mathrm{rank}(B_{2,2}) = 0$ otherwise. Assume $e_{12} \notin \mathcal{E}$, then p_1 and p_2 can be placed arbitrarily, which is a contradiction with statement (2). For robot i $(i \geq 3)$, under Lemma 2, we assume $\mathrm{rank}(B_{i,i}) \neq d$, then robot i only has at most one bearing constraint, such that it can either be randomly placed in $\mathbb{R}^d$ if $\mathrm{card}(\mathcal{P}_i) = 0$, or randomly placed along a line, if either $\mathrm{card}(\mathcal{P}_i) = 1$ or $\forall v_j, v_k \in \mathcal{P}_i, g_{ij} = g_{ik}$. In each of these cases, the framework cannot be unique. Therefore, $\mathrm{rank}(B_{ii}) = d$.

(5) $\Rightarrow$ (4): By the property of block matrices, $\mathrm{rank}(B) \geq \sum_{i=2}^{n} \mathrm{rank}(B_{ii}) = dn - d - 1$. Note that $\mathrm{rank}(B) \leq dn - d - 1$ exists, hence $\mathrm{rank}(B) = dn - d - 1$.

Lemma 3 gives necessary and sufficient conditions to uniquely determine a directed framework with a lower triangular matrix B. Note that the structure of B can be different under distinct labeling rules. Here, we only require that one labeling rule exists, such that B is lower triangular, then Lemma 3 will be applicable immediately.

Infinitesimal bearing rigidity and bearing persistence are generic properties, which are mainly determined by the structure of the underlying graph, rather than the configuration. To highlight this fact, we introduce the following definition:

Definition 3 – Generically Bearing Rigid and Bearing Persistent (GBR-BP) graph: A directed graph $\mathcal{G}$ is GBR-BP in $\mathbb{R}^d$ if there exists at least one configuration $p \in \mathbb{R}^{dn}$ such that $(\mathcal{G}, p)$ in $\mathbb{R}^d$ is IBR and BP.

Lemma 4: Consider a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with a lower triangular bearing Laplacian matrix B. $\mathcal{G}$ is GBR-BP if and only if $\text{card}\,(\mathcal{P}_2) = 1$ and $\text{card}\,(\mathcal{P}_i) \geq 2$, $\forall i \geq 3$.

Proof: *Necessity:* If $\mathcal{G}$ is GBR-BP, there exists a configuration p such that $(\mathcal{G}, p)$ is IBR and BP. Therefore, statement (5) in Lemma 3 should be satisfied. Since $\text{rank}\,(B_{2,2}) = d - 1$, robot 1 should be the parent of robot 2. For $i \geq 3$, $\text{rank}\,(B_{i,i}) = d$ if and only if at least two of $\{g_{ik}\}_{k \in \mathcal{P}_i}$ are not collinear. Thus, $\text{card}\,(\mathcal{P}_i) \geq 2$. *Sufficiency:* If $\text{card}\,(\mathcal{P}_2) = 1$ and $\text{card}\,(\mathcal{P}_i) \geq 2$ exists, we should find a configuration $p = \left[p_1^T, p_2^T, \ldots, p_n^T\right] \in \mathbb{R}^{dn}$, such that $(\mathcal{G}, p)$ is IBR and BP. For p_1, it can be selected randomly. For p_2, because robot 1 is the parent of robot 2, we only need to select $p_2 \neq p_1$, which guarantees $\text{rank}\,(B_{2,2}) = d - 1$. For p_i $(i \geq 3)$, because the position of its parents have been determined, p_i can be selected such that there exist at least two vertices $v_j, v_k \in \mathcal{P}_i$ with $g_{ij} \neq g_{ik}$, which guarantees $\text{rank}\,(B_{i,i}) = d$. In this way, we find one configuration p, such that statement 5 in Lemma 3 is satisfied. Thus, $\mathcal{G}$ is GBR-BP.

Lemma 4 provides an admissible solution to construct directed GBR-BP graphs, and provides the theoretical basis needed to develop construction and re-configuration strategies later. GBR-BP graphs have the following two properties.

Lemma 5: Consider a GBR-BP graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with a lower triangular structure. Add an edge e_{ji} to the graph $\mathcal{G}$, where $v_i, v_j \in \mathcal{V}$ and $j < i$. The resultant graph $\mathcal{G}^+ = (\mathcal{V}, \mathcal{E}^+)$ with $\mathcal{E}^+ = \mathcal{E} \cup \{e_{ji}\}$ is GBR-BP.

Lemma 6: Consider a GBR-BP graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with a lower triangular structure, delete an edge $e_{ki} \in \mathcal{E}$, and add an edge $e_{ji} \notin \mathcal{E}$ with $j < i$. Then, the resultant graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$ with $\mathcal{E}' = (\mathcal{E} \backslash e_{ki}) \cup \{e_{ji}\}$ is GBR-BR.

Lemma 5 and Lemma 6 can be directly derived from Lemma 4, and thus the proofs

are omitted. In other words, Lemma 5 allows us to connect new parent vertices to any vertex v_i $(i \geq 3)$. Lemma 6 allows us to change the parent vertices of any vertex v_i $(i \geq 3)$. This provides us flexibility in adjusting the topology of a robot swarm dynamically, while the bearing rigidity and persistence are guaranteed.

5.3 Problem statement

Based on the concepts of bearing rigidity and bearing persistence, the objective of this chapter is to investigate the construction and reconstruction of self-reconfigurable hierarchical frameworks in a robot swarm. The following three questions will be addressed:

1. Given a swarm of n robots capable of onboard bearing measurements, how can the robots construct a hierarchical and GBR-BP graph?

2. Given the constructed graph, how can the hierarchy and rigidity properties be preserved in self-reconfiguration scenarios, specifically in *merging of frameworks*, *robot departure*, and *splitting of frameworks*?

3. Given the hierarchical frameworks, when coupled with an example control law that makes use of bearing rigidity, how can the robot swarm achieve and reconfigure an arbitrary target formation with moving leaders while preserving the hierarchy and rigidity properties during self-reconfiguration scenarios?

5.4 Framework construction

An important precondition to use Lemma 4 is that the bearing Laplacian of the framework is lower triangular. In this section, we extend Henneberg constructions by introducing the notion of *hierarchy*, which not only guarantees the rigidity and persistence requirement, but also ensures the lower triangular feature of the bearing Laplacian. Our proposed algorithm is inspired by (Trinh et al., 2018), and is defined as follows.

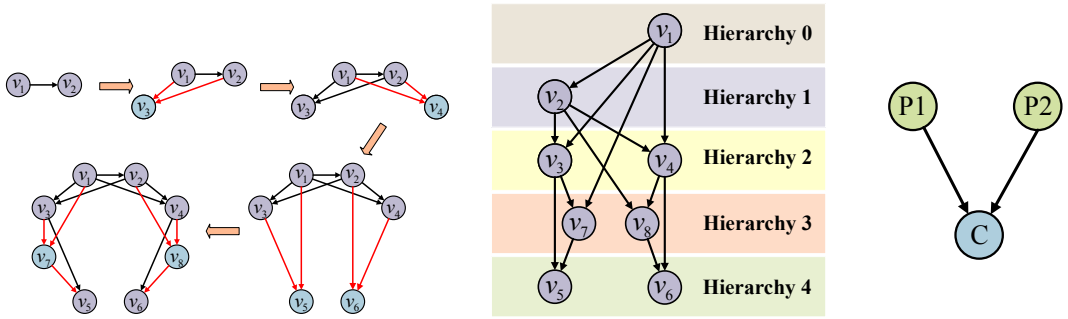
Hierarchical Henneberg construction (HHC). Consider a group of n robots ($n \geq 2$). The first step is to arbitrarily choose two robots in the group as the leader robots, denoted by v_1 and v_2, and add an edge $e_{1,2}$ connecting them. Define the hierarchy $h(v_i)$ of a generic robot v_i as the length of its longest path from v_i to v_1 in the directed graph $\mathcal{G}$. The hierarchy of v_1 and v_2 is 0 and 1, respectively, i.e. $h(v_1) = 0$, $h(v_2) = 1$. In subsequent steps, we utilize one of the following two operations:

1. *Vertex addition*: Add a new vertex v_i to the existing graph, incorporating two directed edges e_{ji} and e_{ki} to two existing vertices v_j and v_k in the graph. Then the hierarchy of vertex v_i is defined as $h(v_i) = \max(h(v_j)), h(v_k))) + 1$.

2. *Edge splitting*: Consider an existing vertex v_k in the graph, which has two parents v_j and v_p in the graph. Remove an edge e_{jk} from the graph and add a new vertex v_i together with three edges e_{ik}, e_{ji} and e_{li}, where vertex v_l is selected such that $h(v_l) \leq h(v_k)$. Then update the hierarchy of v_i as $h(v_i) = \max(h(v_j), h(v_l))+1$ and the hierarchy of v_k as $h(v_k) = \max(h(v_i), h(v_p))+1$.

An example of HHC for a group of eight robots is presented in Fig. 5.3(a). An important feature of HHC is that all the robots, except the two that are arbitrarily selected as leaders, have exactly two parents. Moreover, each follower can form a connection with each of its two parents, forming a *minimal structure* as shown in Fig. 5.3(c). The child receives commands from its parents and obtains its parents' states via communication or sensing, and then uses this information to coordinate with its parents. On the basis of the hierarchical framework, shown in Fig. 5.3(b), the framework can also be viewed as an acyclic tree, with the first of the two leaders as the root.

We define a layer-by-layer labeling rule to verify that the bearing Laplacian matrix of a framework generated by HHC is lower triangular. Let n_l denote the number of vertices with hierarchy l. Vertices with hierarchy 0 are labeled from v_1 to v_{n_0}. Vertices with hierarchy $l \geq 1$ are labeled from $v_{n_{l-1}+1}$ to v_{n_l}. Note that there is no order requirement when labeling vertices with the same hierarchy layer. Based on



(a) (b) (c)

Figure 5.3: An example of HHC for a group of 8 robots. (a) Four steps of construction, with the added vertex and edges in blue and red, respectively. Vertex addition is employed in steps 1–3 and edge splitting is used in step 4. (b) The hierarchy layers of the framework resulting from the construction process in (a), in which the two leaders are on the first and second layers (i.e., 0 and 1). (c) The minimal structure, where P1 and P2 are parents and C is the child. (Reprinted from (Zhang et al., 2023).)

this labeling rule, the bearing Laplacian B can be rewritten as

$$B = \begin{bmatrix} \mathbf{0} & & & & & \mathbf{0} \\ B_{2,1} & B_{2,2} & & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ \mathbf{0} & -B_{ij} & \mathbf{0} & -B_{ik} & B_{ii} & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{5.7}$$

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ generated via Henneberg construction is called a *Laman graph* (Laman, 1970). It was proved in citezhao2017laman that an *undirected Laman graph* is generically bearing rigid. Here, we further shows that a *directed Laman graph* is GBR-BP.

Theorem 1. A graph $\mathcal{G}$, generated by HHC, is GBR-BP.

Proof. Following Lemma 4, $\mathcal{G}$ is GBR-BP if and only if $\text{card}\,(\mathcal{P}_2) = 1$ and $\text{card}\,(\mathcal{P}_i) \geq 2$, $\forall i \geq 3$. Denote the graph consisting of n vertices as $\mathcal{G}_n = (\mathcal{V}_n, \mathcal{E}_n)$.

Firstly, we consider the case of $n = 2$. $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ is defined as $\mathcal{V}_2 = \{v_1, v_2\}$ and $\mathcal{E}_2 = \{e_{1,2}\}$. Note that the bearing Laplacian matrix of $\mathcal{G}_2$ is lower triangular, and $\text{card}\,(\mathcal{P}_2) = 1$. Therefore, the claim is true for $n = 2$.

Secondly, suppose that the claim is true for $2 \leq l \leq n-1$. Now, we consider the case of $l = n$, i.e., a new vertex v_n will be added to $\mathcal{G}_{n-1}$. According to HHC, there are the following two cases.

Vertex addition: Select two distinct vertices v_j and v_k from $\mathcal{G}_{n-1}$. We add edges e_{jn} and e_{kn}. It is trivial to verify that the bearing Laplacian matrix is still lower-triangular, as $j, k < n$. Moreover, given that $\mathcal{G}_{n-1}$ is GBR-BP and $\text{card}\,(\mathcal{P}_n) = 2$, then $\mathcal{G}_n$ is GBR-BP under Lemma 4.

Edge splitting: We select three vertices v_k, v_j, and v_l from $\mathcal{G}_{n-1}$, according to the requirements specified in the operation description. Then the new graph is given by $\mathcal{G}_n = (\mathcal{V}_n, \mathcal{E}_n)$, where $\mathcal{V}_n = \mathcal{V}_{n-1} \cup \{v_n\}$ and $\mathcal{E}' = \mathcal{E} \setminus e_{jk} \cup \{e_{jn}, e_{ln}, e_{nk}\}$. We relabel the vertices according to our labeling rule, such that the bearing Laplacian matrix is verified to be lower triangular. We verify that $\text{card}\,(\mathcal{P}_i) = 2$ is guaranteed $\forall 3 \leq i \leq n$. It follows from Lemma 3 that $\mathcal{G}_n$ is GBR-BP.

The constructed framework can be considered centralized, in the sense that two leaders have the ability to indirectly control the whole swarm. It can also be considered decentralized, because each follower only needs the local information associated with its parents. This reflects the targeted Mergeable Nervous Systems concept (Mathews et al., 2017), supporting parallel processing even in large-scale robot swarms.

Our proposed construction process contributes frameworks that exhibit the following key properties:

1. The framework benefits from rigidity and hierarchy. These two features provide a theoretical basis to predict the motion of each robot, and can facilitate human operators controlling the behavior of the swarm.

2. On the basis of rigidity and hierarchy, we can dynamically change the size of the framework via the framework reconstruction strategies proposed in Sec. V. This flexibility of swarm size enables regulation of frameworks according to

Algorithm 1 Constructing a hierarchical GBR-BP graph of n ($n \geq 2$) robots in $\mathbb{R}^d$ ($d \geq 2$)

$i \leftarrow 0$;
Choose arbitrarily two robots from the swarm to define as leaders v_1 and v_2;
$\mathcal{G} \leftarrow$ add vertices v_1 and v_2, and edge $e_{2,1} = (v_2, v_1)$;
$i \leftarrow i + 2$;
$h(v_1) \leftarrow 0$, $h(v_2) \leftarrow 1$;
while $i \leq n$ **do**
  $i \leftarrow i + 1$;
  if Vertex addition is performed **then**
    Choose arbitrarily two robots to define as v_j and v_k from $\mathcal{G}$;
    $\mathcal{G} \leftarrow$ Add a vertex v_i and two edges e_{ij} and e_{ik};
    $h(v_i) \leftarrow \max(h(v_j), h(v_k))) + 1$;
  else if Edge splitting is performed **then**
    Choose arbitrarily one robot to define as v_k from $\mathcal{G}$, which has two parent robots v_j and v_p;
    Choose arbitrarily one robot to define as v_l from $\mathcal{G}$, satisfying $h(v_l) \leq h(v_k)$;
    $\mathcal{G} \leftarrow$ Remove edge e_{kj}, add one robot v_i and three edges e_{ki}, e_{ij} and e_{il};
    $h(v_i) \leftarrow \max(h(v_j), h(v_l)) + 1$;
    $h(v_k) \leftarrow \max(h(v_i), h(v_p)) + 1$;
  end if
end while
return $\mathcal{G}$;

task requirements and environment constraints.

3. The framework has no reliance on external position or distance measurements, instead using only bearing measurement. When coupled with an example control law, e.g., for reactive formation control with moving leaders, the robot swarm can achieve self-organized formations using only relative bearing measurement and local interactions, as shown in Sec. VI.

Remark 1: The concept of hierarchy has been reflected in the field of bearing-based formation control, in formation maneuvering (Trinh and Ahn, 2021) and Henneberg Construction (Trinh et al., 2018). However, our research differs from these and contributes in two major ways. 1) Bearing rigidity was not discussed in (Trinh and Ahn, 2021). We address this gap by analyzing the rigidity of hierarchical frameworks based on the notion of bearing persistence. 2) Hierarchy was introduced as a concept in (Trinh et al., 2018), but not fully investigated. We expand on the existing work and propose self-reconfiguration algorithms on the basis of hierarchical structures.

5.5 Framework reconstruction

In this section, we address the problem of framework reconstruction. The case of adding a new robot can be addressed directly by the vertex addition and edge splitting operations introduced in Sec. IV. The remaining cases of reconstruction are more challenging and require explication. Accordingly, in this section, we address the cases of *merging frameworks*, *robot departure*, and *splitting frameworks*.

5.5.1 Merging frameworks

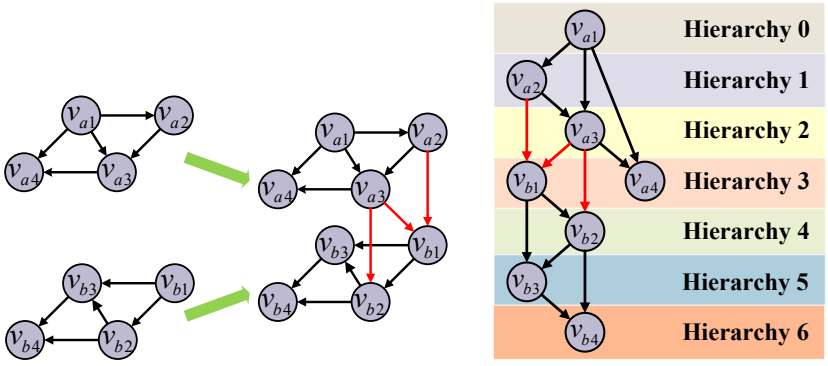
This section concerns the problem of merging two frameworks. A merging strategy for undirected frameworks that considers maintenance of bearing rigidity has been proposed in (Trinh et al., 2019). We build upon (Trinh et al., 2019) by extending to the case of directed graphs and maintenance of the hierarchical structure and bearing persistence.

First, consider two directed hierarchical frameworks: $(\mathcal{G}_a, p_a)$ with n_a robots, and $(\mathcal{G}_b, p_b)$ with n_b robots, where $n_a, n_b \geq 2$. Fundamentally, we need to find the minimum number of new edges to be added, in order to maintain bearing rigidity and persistence.

Theorem 2. Consider two graphs $\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a)$ and $\mathcal{G}_b = (\mathcal{V}_b, \mathcal{E}_b)$, generated by HHC. Denote two leaders of framework $(\mathcal{G}_b, p_b)$ as v_{b_1} and v_{b_2} and perform the following sequence of operations: 1) Select two vertices $v_{a_i}, v_{a_j} \in \mathcal{V}_a$; 2) Add three edges $e_1 = (v_{a_i}, v_{b_1})$, $e_2 = \left(v_{a_j}, v_{b_1}\right)$, and $e_3 = \left(v_{a_j}, v_{b_2}\right)$. The resulting post-merged graph $\bar{\mathcal{G}} = \left\{\bar{\mathcal{V}}, \bar{\mathcal{E}}\right\}$ defined by $\bar{\mathcal{V}} = \mathcal{V}_a \cup \mathcal{V}_b$ and $\bar{\mathcal{E}} = \mathcal{E}_a \cup \mathcal{E}_b \cup \{e_1, e_2, e_3\}$ is GBR-BP.

Proof. We add two edges to v_{b_1} and one edge to v_{b_2}, which results in card $(\mathcal{P}_i) = 2$ for all $3 \leq i \leq n_a + n_b$. We can therefore employ Lemma 4 to verify that the post-merged graph is GBR-BP.

Theorem 2 implies that, after adding three edges, the post-merged graph is GBR-BP. In addition, the hierarchical structure of the framework is preserved. After the merging operation, the hierarchy of robots in the framework $(\mathcal{G}_B, p_B)$ should be updated as $h\left(v_{B_i}\right) \leftarrow h\left(v_{B_i}\right) + \max\left(h\left(v_{a_i}\right), \left(v_{a_j}\right)\right) + 1$. An example of merging two frameworks is given in Fig. 5.4.



(a) (b)

Figure 5.4: An example of merging two frameworks. (a) A cube framework is constructed by merging two square frameworks (added edges in red). (b) The hierarchy layers of the post-merged framework. (Reprinted from (Zhang et al., 2023).)

Motivated by Theorem 2, we extend the merging strategy to the case of m graphs, as summarized in Algorithm 2. It should be noted that, using Algorithm 2, merging processes can be performed in series or in parallel. In the case of m graphs, the merging processes can be grouped into a minimum of 1 groups (i.e., all in parallel) and a maximum of $m-1$ groups (i.e., all in series), such that the time complexity will be between $\mathcal{O}(1)$ and $\mathcal{O}(m)$. Therefore, with the proposed merging operation, we can accelerate the construction process of large-scale robot swarms. For instance, we can construct various hierarchical and rigid frameworks simultaneously via Algorithm 1, and at the same time, Algorithm 2 can be utilized to merge these frameworks, achieving a faster self-organization process via parallelization.

Algorithm 2 Merging m GBR-BP graphs $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m$ into one GBR-BP graph $\bar{\mathcal{G}}$ in $\mathbb{R}^d$ $(d \geq 2)$

$\bar{\mathcal{G}} \leftarrow \mathcal{G}_1$;
for $k = 2 \rightarrow m$ **do**
  Select two vertices v_i and v_j from $\bar{\mathcal{G}}_{k-1}$;
  Select leader vertices v_{k_1} and v_{k_2} from $\mathcal{G}_k$;
  Add edges $e_1 = (v_i, v_{k_1})$, $e_2 = (v_j, v_{k_2})$, and $e_3 = (v_j, v_{k_2})$;
  $\bar{\mathcal{G}} = \left(\bar{\mathcal{V}}, \bar{\mathcal{E}}\right)$, where $\bar{\mathcal{V}} \leftarrow \bar{\mathcal{V}} \cup \mathcal{V}_k$ and $\bar{\mathcal{E}} \leftarrow \bar{\mathcal{E}} \cup \mathcal{E}_k \cup \{e_1, e_2, e_3\}$;
  Update the hierarchy of vertices in $\mathcal{G}_k$ as $h\left(v_{k_i}\right) \leftarrow h\left(v_{k_i}\right) + \max\left(h\left(v_i\right), \left(v_j\right)\right) + 1$;
end for
return $\bar{\mathcal{G}}$;

5.5.2 Robot departure

In this subsection, we consider the removal of a robot from the framework. According to whether a robot has a child or not, the robots in the swarm can be classified into two categories: outer node (i.e., no child) and inner node (i.e., at least one child). We consider the robot departure problem in both cases.

Case 1: Removal of an outer node.

We first consider the case with an outer node robot, e.g., v_7 and v_8 in Fig. 5.5(a). Consider a directed hierarchical framework $(\mathcal{G}, p)$ with n robots $(n \geq 2)$. We assume

that the vertex v_n is an outer node and its parent vertices are relabelled as v_i and v_j.

Theorem 3: Given a GBR-BP graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ generated by HHC, remove an outer node vertex v_n and two associated edges e_{in} and e_{jn}, the graph $\mathcal{G}^- = (\mathcal{V}^-, \mathcal{E}^-)$ defined by $\mathcal{V}^- = \mathcal{V} \backslash \{v_n\}$ and $\mathcal{E}^- = \mathcal{E} \backslash \{e_{in}, e_{jn}\}$ is GBR-BP.

The proof of Theorem 3 is omitted here, because the removal of an outer node is an inverse operation of "vertex addition" in HHC. Lemma 3 can be used to verify the rigidity of the framework after the deletion of an outer node.

Case 2: Removal of an inner node.

When an inner node robot leaves the framework (e.g., v_4 in Fig. 5.5(a)), the rigidity of the framework is destroyed and needs to be repaired. To repair the rigidity and maintain the hierarchical structure, the following corollary can be derived.

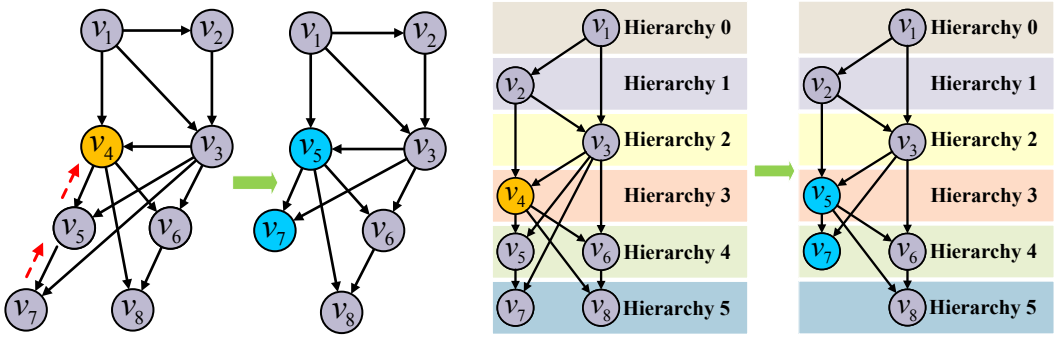


Figure 5.5: An example of robot departure when using Algorithm 3. Step 1: Vertex v_4 (yellow) is removed from the framework. Step 2: Vertex v_5 (blue) is shifted to replace v_4 and vertex v_7 (blue) is shifted to replace v_5. (a) and (b) show the frameworks and the corresponding hierarchy layers, respectively, of the two steps. (Reprinted from (Zhang et al., 2023).)

Corollary 1: Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ generated by HHC, if an inner vertex v_k leaves the framework, let its position in the framework (including hierarchy and connected edges) be inherited by the one of its children vertices v_m that has the highest hierarchy in $\mathcal{C}_k$. In other words, $\forall v_j \in \mathcal{C}_k, h(v_m) \geq h(v_j)$. If v_m is an outer node,

Theorem 3 yields that the resultant graph after performing the *inheritance* operation is GBR-BP. If v_m is an inner node, we can continue performing the inheritance operation until an outer node is reached.

The strategy stated in Corollary 1 is summarized in Algorithm 3 and an example is given in Fig. 5.5. With the help of our proposed algorithm, we can remove any robot from the framework without destroying rigidity, persistence, and hierarchical architecture. One advantage of the proposed method is that only *local* information is required to perform the inheritance operation. The time complexity of our proposed robot departure algorithm can be calculated as $\mathcal{O}(n)$. In contrast to our approach, existing methods such as (Hou and Yu, 2016) require an optimal repairing solution from the *global* perspective to find the necessary edges to maintain the rigidity.

Algorithm 3 Removal of vertex v_k from GBR-BP graph $\mathcal{G}$ in $\mathbb{R}^d$ $(d \geq 2)$

$\mathcal{V} \leftarrow \mathcal{V} \backslash \{v_k\}, \mathcal{E} \leftarrow \mathcal{E} \backslash \{e_{ij} | v_i \text{ or } v_j = v_k\}$;
while v_k is not an outer node vertex **do**
  Select vertex $v_m \in \mathcal{C}_k$, such that $\forall v_j \in C_k, h(v_m) \geq h(v_j)$;
  $\mathcal{G} \leftarrow$ Remove the edges associated with v_m;
  $\mathcal{G} \leftarrow$ Add edges from vertices in $\mathcal{P}_k$ to v_m;
  $\mathcal{G} \leftarrow$ Add edges from v_m to $v_j \in \mathcal{C}_k \backslash v_m$;
  $h(v_m) \leftarrow h(v_k)$;
  $v_k \leftarrow v_m$;
end while
return $\mathcal{G}$;

Remark 2: Note that for the case of multiple robots being removed at the same time, it might not always be possible to employ the inheriting operation as introduced in Corollary 1, because the hierarchical structure would not always be maintained. To reconstruct frameworks under such a scenario, based on Lemma 4, each follower should possess at least two parent vertices. For this reason, Algorithm 4 presents a protocol for the followers of removed robots to select new parents under the hierarchy constraint.

Algorithm 4 Removal of k ($k \geq 2$) vertices from GBR-BP graph $\mathcal{G}$ in $\mathbb{R}^d$ ($d \geq 2$)

$\mathcal{V} \leftarrow \mathcal{V} \backslash \{v_{m_1}, \ldots, v_{m_k}\}, \mathcal{E} \leftarrow \mathcal{E} \backslash \{e_{ij} | v_i \text{ or } v_j \notin \mathcal{V}\}$;
Select two vertices $v_{l_1}, v_{l_2} \in \mathcal{V}$ such that $h(v_{l_1}) \leq h(v_{l_2}) \leq h(v_i), \forall v_i \in \mathcal{V}$;
$h(v_{l_2}) \leftarrow h(v_{l_1}) + 1$;
$h(v_i) \leftarrow h(v_i) + 2, \forall v_i \in \mathcal{V} \backslash \{v_{l_1}, v_{l_2}\}$;
for $v_i \in \mathcal{V} \backslash \{v_{l_1}, v_{l_2}\}$ **do**
  while card $\{\mathcal{P}_i\} < 2$ **do**
    $\mathcal{G} \leftarrow$ add new edge e_{qi}, where $v_q \in \mathcal{V}$ is chosen such that $v_q \notin \mathcal{P}_i$ and $h(v_q) < h(v_i)$;
  end while
  Update $h(v_i)$ according to the hierarchy of its parents;
end for
return $\mathcal{G}$;

5.5.3 Splitting frameworks

In this subsection, we consider the case where a framework with at least four robots is split into several disjoint sub-frameworks, each consisting of at least two robots. Similar to the merging operation, the main difficulty of the splitting operation is preservation of the bearing rigidity, persistence, and hierarchy of the sub-frameworks after splitting. Note that the splitting operation can be considered a generalized extension of robot departure. Without loss of generality, we first consider the strategy for splitting one framework into two sub-frameworks.

We use a special graph called Z-link, originally proposed in (Olfati-Saber and Murray, 2002) and employed in (Carboni et al., 2014) for undirected graphs. We extend this existing research to directed Z-links. We denote Z-link by $Z = (\mathcal{V}_Z, \mathcal{E}_Z)$, where $|\mathcal{V}_Z| = 4$ and $|\mathcal{E}_Z| = 3$, as shown in Fig. 5.6. The following definition determines the existence of a Z-link in a graph $\mathcal{G}$.

Definition 4 – Z-link: Consider a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Two disjoint subgraphs $\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a)$ and $\mathcal{G}_b = (\mathcal{V}_b, \mathcal{E}_b)$ are said to be connected via a Z-link if the following two conditions hold.

1. There exist four distinct vertices $v_{a1}, v_{a2} \in \mathcal{V}_a$ and $v_{b1}, v_{b2} \in \mathcal{V}_b$, such that the graph among these four vertices is a Z-link.

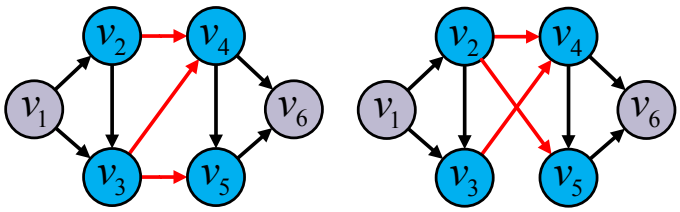


Figure 5.6: Two possible configurations of directed Z-links (marked in red), based on undirected Z-links (Carboni et al., 2014). (Reprinted from (Zhang et al., 2023).)

2. $\mathcal{V}_a \cup \mathcal{V}_b = \mathcal{V}$, $\mathcal{V}_a \cap \mathcal{V}_b = \varnothing$, $\mathcal{E}_a \cap \mathcal{E}_b = \varnothing$, and $\mathcal{E} = \mathcal{E}_a \cup \mathcal{E}_b \cup \mathcal{E}_Z$.

Theorem 4: Given a GBR-BP graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, let $\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a)$ and $\mathcal{G}_b = (\mathcal{V}_b, \mathcal{E}_b)$ be two disjoint subgraphs of $\mathcal{G}$, which are connected via a Z-link. Then, $\mathcal{G}_a$ is GBR-BP $\Leftrightarrow$ $\mathcal{G}_b$ is GBR-BP.

Proof: We only show that $\mathcal{G}_a$ is GBR-BP $\Rightarrow$ $\mathcal{G}_b$ is GBR-BP, because the reverse is the same.

Given that $\mathcal{G}$ is GBR-BP, there exists a configuration $p = \left[p_1^T, \ldots, p_n^T\right]^T \in \mathbb{R}^{dn}$, such that $(\mathcal{G}, p)$ is IBR and BP. Let $p_a = \left[p_1^T, \ldots, p_{n_a}^T\right]^T \in \mathbb{R}^{dn_a}$ and $p_b = \left[p_{n_a+1}^T, \ldots, p_n^T\right]^T \in \mathbb{R}^{d(n-n_a)}$. Let B_b be the bearing Laplacian matrix of $(\mathcal{G}_b, p_b)$.

Without loss of generality, we assume that $\mathcal{V}_a \cap \mathcal{V}_Z = \{v_{a1}, v_{a2}\}$ and $V_b \cap V_Z = \{v_{n_a+1}, v_{n_a+2}\}$. We add edge $e_{(n_a+1)(n_a+2)}$ to the graph $\mathcal{G}$. Then, the resultant graph $\mathcal{G}^+ = \left(\mathcal{V}, \mathcal{E} \cup e_{(n_a+1)(n_a+2)}\right)$ is still GBR-BP under Lemma 5. Denote B^+ as the bearing Laplacian matrix of $(\mathcal{G}^+, p)$.

We augment $\mathcal{G}_a$ to $\mathcal{G}_a^+ = (\mathcal{V}_a^+, \mathcal{E}_a^+)$, defined by $\mathcal{V}_a^+ = \mathcal{V}_a \cup \{v_{n_a+1}, v_{n_a+2}\}$ and $\mathcal{E}_a^+ = \mathcal{E}_a \cup \mathcal{E}_Z \cup e_{(n_a+1)(n_a+2)}$. Denote B_a^+ as the bearing Laplacian matrix of $(\mathcal{G}_a^+, p_a^+)$, where $p_a^+ = \left[p_a^T, p_{n_a+1}^T, p_{n_a+2}^T\right]^T \in \mathbb{R}^{d(n_a+2)}$.

As a result, we can write the bearing rigidity matrix B^+ as

$$B^+ = \left[\begin{array}{c:c} B_a^+ & \mathbf{0} \\ \hdashline \mathbf{0} & \mathbf{0} \end{array}\right] + \left[\begin{array}{c:c} \mathbf{0} & \mathbf{0} \\ \hdashline \mathbf{0} & B_b \end{array}\right] \tag{5.8}$$

Consider equation $B_b q = \mathbf{0}$. If $\mathcal{G}_b$ is not GBR-BP, then there exists a configuration $q_b = \left[q_{n_a+1}^T, \ldots, q_n^T\right] \in \mathbb{R}^{d(n-n_a)}$ such that $q_{n_a+1} = p_{n_a+1}$ and $q_{n_a+2} = p_{n_a+2}$, but

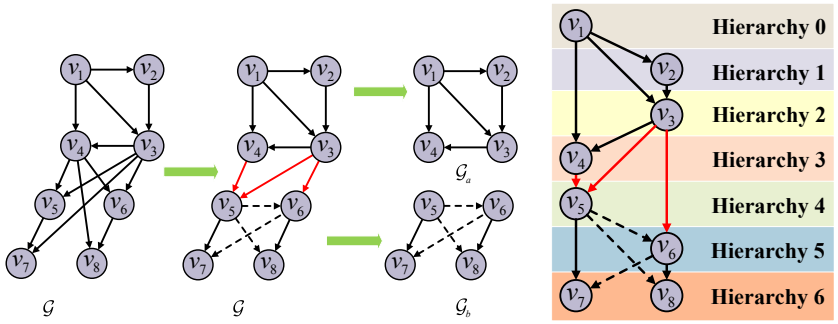
$q_i \neq p_i$, $\forall i \in \{n_a + 3, \ldots, n\}$.

Let $q^{'} = \left[p_1^T, \ldots, p_{n_a+1}^T, p_{n_a+2}^T, q_{n_a+3} \ldots, q_n\right]^T$. Eq. (5.8) yields $B^+ q^{'} = 0$. Note that $q^{'} \notin \text{span}\{\mathbf{1}_n \otimes \mathbf{I}_d, p\}$, i.e., $\mathcal{G}^+$ is not GBR-BP, which is a contradiction. Therefore, $\mathcal{G}_b$ is verified to be GBR-BP. ■

Theorem 4 indicates that, for any GBR-BP graph, if there exists a Z-link connecting two disjoint subgraphs $\mathcal{G}_a$ and $\mathcal{G}_b$, one of which is guaranteed to be GBR-BP, then the other subgraph is also GBR-BP. This lemma leads us to develop the following splitting strategy: we firstly find a GBR-BP subgraph $\mathcal{G}_a$, and secondly construct a Z-link between $\mathcal{G}_a$ and $\mathcal{G}_b$. After the removal of the Z-link edges, we obtain two GBR-BP subgraphs. We now present our 2-step algorithm to split the framework, exploiting the triangularity in Eq. (5.7).

Step 1: Find a GBR-BP subgraph $\mathcal{G}_a$.

Given a framework generated by HHC with n robots, the Bearing Laplacian submatrix of the first n_a robots always satisfies a triangular structure (cf. the triangularity in Eq. (5.7). Therefore, we can verify $\mathcal{G}_a$ as GBR-BP according to the first n_a robots, as stated in the following Theorem.



(a) (b)

Figure 5.7: An example of splitting one framework $\mathcal{G}$ into two sub-frameworks $\mathcal{G}_a$ and $\mathcal{G}_b$ when using Algorithm 5. (a) First, vertices v_5 and v_6 are chosen as the leaders for $\mathcal{G}_b$. Second, a Z-link (marked in red) is constructed between the two sub-frameworks (newly added edges are marked as dashed arrows). Third, the Z-link is removed, resulting in two separate frameworks. (b) The corresponding hierarchy layers after Z-link construction (before Z-link removal). (Reprinted from (Zhang et al., 2023).)

Lemma 7: Given a GBR-BP graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ generated by HHC, let $\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a)$ represent a subgraph describing the interactions corresponding to first n_a vertices, i.e., $\mathcal{V}_a = \{v_1, \ldots, v_{n_a}\}$. Then, $\mathcal{G}_a$ is GBR-BP.

Proof: Denote B as the bearing rigidity Laplacian matrix of $(\mathcal{G}, p)$, where $p \in \mathbb{R}^{nd}$ is a configuration. Then B can be partitioned as

$$B = \begin{bmatrix} B_a & \mathbf{0} \\ B_{b1} & B_{b2} \end{bmatrix} \tag{5.9}$$

where $B_a \in \mathbb{R}^{dn_a \times dn_a}$ denotes the bearing Laplacian matrix for the first n_a vertices. Then we can apply statement (5) of Lemma 3 to verify the rank of matrices on diagonal of B_a, which shows that $\mathcal{G}_a$ is GBR-BP. ■

Step 2: Construct a Z-link between two subgraphs.

Let $\mathcal{G}_b = (\mathcal{V}_b, \mathcal{E}_b)$ represent the interactions among the remaining vertices, i.e., $\mathcal{V}_b = \{v_{n_a+1}, \ldots, v_n\}$. Corresponding to Definition 4, Z-link construction comprises the following two parts.

1. Let $\mathcal{P}_{n_a+1} = \{v_{p1}, v_{p2}\}$. Remove the ingoing edges of v_{n_a+2}, then add edges $e_{(n_a+1)(n_a+2)}$ and $e_{p_1(n_a+2)}$.

2. $\forall v_i \in V_b \backslash \{v_{n_a+1}, v_{n_a+2}\}$, if its parent $v_j \in V_a$, then remove e_{ji}. To preserve rigidity, add new edge e_{ki}, where new parent $v_k \in \mathcal{V}_b$ is chosen such that $v_k \notin \mathcal{P}_i$ and $h(v_k) < h(v_i)$.

Here, Lemma 6 is repeatedly employed to satisfy Definition 4, therefore the resultant graph is still GBR-BP.

Finally, we can use the above splitting strategy for the case of m graphs, as summarized in Algorithm 5. Similar to the merging process, according to the for-loop structure of the splitting algorithm, the time complexity can be calculated as $\mathcal{O}(mn)$. Note that multiple splitting processes can happen in parallel. This is because the robots have been divided into m subsets $\mathcal{V}_k = \left\{v_{n_{k-1}+1}, \ldots, v_{n_{k-1}+n_k}\right\}$, and each follower is required to change its parent vertex such that $\forall v_i \in \mathcal{V}_k, \mathcal{P}_i \subset \mathcal{V}_k$. For instance, we can simultaneously apply the splitting algorithm to all subsets m; then

Algorithm 5 Splitting one GBR-BP graph $\mathcal{G}$ into m GBR-BP graphs: $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m$ in $\mathbb{R}^d$ $(d \geq 2)$.

$n_0 = 0$;
for $k = 1 \to m$ **do**
  $\mathcal{V}_k \leftarrow \left\{v_{n_{k-1}+1}, \ldots, v_{n_{k-1}+n_k}\right\}$;
  Select vertex $v_{n_{k-1}+1}$ as the first leader of $\mathcal{G}_k$, and denote its parents as $v_{p_k^1}$ and $v_{p_k^2}$;
  Select vertex $v_{n_{k-1}+2}$ as the second leader of $\mathcal{G}_k$;
  $\mathcal{G} \leftarrow$ Remove the ingoing edges of $v_{n_{k-1}+2}$;
  $\mathcal{G} \leftarrow$ Construct Z-link by adding two ingoing edges to the second leader: $e_{k_1} = \left(v_{n_{k-1}+1}, v_{n_{k-1}+2}\right)$ and $e_{k_2} = \left(v_{p_k^1}, v_{n_{k-1}+2}\right)$;
  $h\left(v_{n_{k-1}+2}\right) \leftarrow h\left(v_{n_{k-1}+1}\right) + 1$;
  $h\left(v_i\right) \leftarrow h\left(v_i\right) + 2, \forall i \in \left\{n_{k-1}+3, \ldots, n_{k-1}+n_k\right\}$;
  for $i = n_{k-1} + 3$ to $n_{k-1} + n_k$ **do**
    for $v_j \in \mathcal{P}_i$ **do**
      if $v_j \notin \mathcal{V}_k$ **then**
        $\mathcal{G} \leftarrow$ remove edge e_{ji};
        $\mathcal{G} \leftarrow$ add new edge e_{qi}, where $v_q \in \mathcal{V}_k$ is chosen such that $v_q \notin \mathcal{P}_i$ and $h\left(v_q\right) < h\left(v_i\right)$;
      end if
    end for
    Update $h\left(v_i\right)$ according to the hierarchy of its parents;
  end for
  $\mathcal{E}_k \leftarrow \left\{e_{ij} \subset \mathcal{E} | v_i, v_j \in \mathcal{V}_k\right\}$;
end for
$\mathcal{G} \leftarrow$ Remove all constructed Z-links by deleting two ingoing edges of $v_{n_{k-1}+1}$ and edge e_{k_2};
return $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m$;

the time complexity will be determined by the maximum subset size, so it will be reduced to $\mathcal{O}(\max\{n_k\})$. Therefore, in practice, the time complexity will often be lower than $\mathcal{O}(mn)$. A splitting example is presented in Fig. 5.7. Note that Algorithm 5 can split the framework into at least two subgraphs with arbitrary size no less than 2, which provides flexibility in managing the size of the framework.

5.6 Validation with an example control law

In this section, to demonstrate and validate our theoretical results, we couple our proposed hierarchical frameworks with an example formation control law and then apply them to four example scenarios of reactive formation control in an aerial robot swarm with moving leaders. In the first scenario, we establish a target formation based on our proposed hierarchical framework and validate Theorem 1. In the second, we merge two formations under Algorithm 2 and validate Theorem 2. Third, we show robot departure from a formation under Algorithm 3 and validate Theorem 3 and Corollary 1. Fourth, we split a formation under Algorithm 5 and validate Theorem 4 and Lemma 7.

We consider a group of n mobile robots moving in $\mathbb{R}^d$ ($d \geq 2$), the model of which is described by a single integrator $\dot{p}_i = u_i$, where $p_i \in \mathbb{R}^d$ is the inertial position of ith robot and $u_i \in \mathbb{R}^d$ is the control input. The main purpose of this section is to validate our proposed construction and reconstruction algorithms, when coupled with an example control law, using experimental results in simulation. Therefore, only a single-integrator model is considered. (For further results on formation control with two-leader directed frameworks, please refer to (Zhang et al., 2022).)

5.6.1 Example scenario 1: Achieving the desired formation

Consider a swarm with n robots with a hierarchical topology, characterized by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ generated by Algorithm 1. The robots are located at $p_1, \ldots, p_n$ in $\mathbb{R}^d$, and each robot does not know the global position p_i but can sense the bearing vectors with regard to its parent robots, i.e., $\{g_{ij} | v_j \in \mathcal{P}_i\}$. We assume that the positions of the neighboring robots do not coincide, i.e., $\forall e_{ij} \in \mathcal{E}$ and $t \geq 0$,

$p_i(t) \neq p_j(t)$, which guarantees the bearing vectors to be well-defined.

In our hierarchical framework, two robots denoted by v_1 and v_2 are chosen as the leaders, while the others are followers. Given a set of feasible desired bearings $\left\{g_{ij}^* | e_{ji} \in \mathcal{E}\right\}$ among the robots[3], the desired robot formation is uniquely characterized, but for rigid translation and scaling. This last ambiguity of the graph is resolved by fixing the position of the first robot, and the distance of the first two robots. Define $p^*(t)$ as the vector of desired position of the robots over time, and $d_{ij}^*(t) = ||p_i^*(t) - p_j^*(t)||$ as the distance between the desired position of robot i and of robot j. The relationship between $p^*(t)$ and $\left\{g_{ij}^* | e_{ji} \in \mathcal{E}\right\}$ is characterized by Lemma 1 in (Trinh et al., 2018).

According to Lemma 1 in (Trinh et al., 2018), given the position of two leaders, a framework constructed by HHC can be uniquely determined. Moreover, the desired translational and scaling maneuvers of the formation are uniquely determined by the reference motion of the first two "leader robots." This inherent property also shows the possibility of using *centralized* decision-making behaviors with our self-organized hierarchical frameworks, because we can control the translation and scale of the formation via two leader robots, which reduces the complexity of formation management.

In this chapter, for the sake of simplicity, we do not consider the motion control of leaders, and we assume that the two leaders move along predefined trajectories, i.e., $p_1(t) = p_1^*(t)$ and $p_2(t) = p_1(t) - d_{2,1}^*(t) g_{2,1}^*$, at all time $t > 0$ [4]. In order to drive the followers to achieve the desired formation, the following bearing-only formation control law for robot v_i $(i \geq 3)$ is used

$$\dot{p}_i = u_i = -c\left(P_{g_{ij}} g_{ij}^* + P_{g_{ik}} g_{ik}^*\right) + \dot{p}_i^* \tag{5.10}$$

where c is a positive constant to be tuned and $\dot{p}_i^*$ is a feedforward term given below.

$$\dot{p}_i^*(t) = \left(P_{g_{ij}^*} + P_{g_{ik}^*}\right)^{-1} \left(P_{g_{ij}^*} \dot{p}_j^*(t) + P_{g_{ik}^*} \dot{p}_k^*(t)\right) \tag{5.11}$$

[3]The feasibility conditions are specified in Assumption 2 in (Trinh et al., 2018).

[4]Note that this assumption is not limitative and that all the results of this chapter hold true by using suitable control laws for the two leaders ensuring convergence to the desired trajectories.

and each agent can compute $\dot{p}_i^*(t)$ by receiving $\dot{p}_j^*(t)$ and $\dot{p}_k^*(t)$ from its parents.

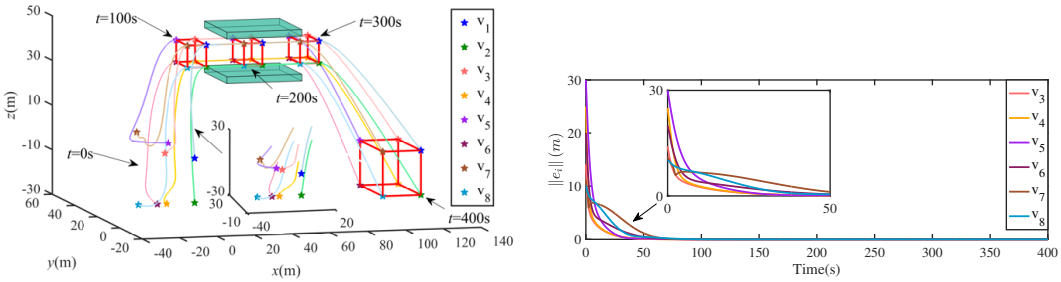
$$\begin{cases} p_1(t) = [0.3t, 40\sin(\pi t/200), 40\sin(\pi t/200)]^T, d_{2,1}^*(t) = 20 - 10\sin(\pi t/200), t \leq 100 \\ p_1(t) = [0.3t, 40, 40]^T, d_{2,1}^*(t) = 10, 100 \leq t \leq 300 \\ p_1(t) = [0.3t, 40\sin(\pi(t-200)/200), 40\sin(\pi(t-200)/200)]^T, d_{2,1}^*(t) \\ \quad = 20 - 10\sin(\pi(t-200)/200), t \geq 300 \end{cases} \tag{5.12}$$

Remark 3: The control law (5.10) is inspired by (Trinh et al., 2018), in which the formation is static. To extend this zero-velocity control law to moving formations, we introduce the feedforward term $\dot{p}_i^*(t)$ to guarantee zero steady-state error. Note that transmission and computation of feedforward terms through the hierarchy is not instantaneous and will introduce delays. There are several approaches to handle such delays (e.g., sufficient preview of the reference signal, or relaxing the perfect tracking requirement and proving ISS-like properties assuming a purely reactive control law). However, such analyses are nontrivial and are beyond the scope of this chapter. The specific control law used (5.10) is just an example to demonstrate the effectiveness of our proposed framework; any control law that makes use of bearing rigidity could in principle be coupled with our frameworks.

By employing a similar stability analysis as shown in Theorem 1 of (Trinh et al., 2018), we can also demonstrate that the formation tracking error $e_i(t) = p(t) - p^*(t)$ asymptotically converges to zero using the control law (5.10). Note that the implementation of control law (5.10) requires only local measurement and local communication from parents to children, which supports the *decentralized* coordination targeted in a reactive swarm approach.

Remark 4: The only parameter to be tuned in Eq. (5.10) is the control gain c. c should be positive, and an increase of c will speed up the formation achievement, but will also result in a larger velocity amplitude. Therefore, the trade-off between convergence speed and velocity amplitude should be considered when defining c.

A simulation example is shown in Fig. 5.8. We consider a group of eight robots



(a) (b)

Figure 5.8: Simulation results in Example scenario 1: Achieving the desired formation. (a) Trajectories of robots. (b) Formation tracking errors of robots. (Reprinted from (Zhang et al., 2023).)

with hierarchical framework shown in Fig. 5.3(a), and the target formation is a cube. The motion of leader v_1 and time-varying distance $d^*_{2,1}$ are shown in Eq. (5.12). The controller parameter is chosen as $c = 5$. Fig. 5.8(a) depicts the trajectory of eight robots. As can be seen, the target formation can be achieved while the centroid and scale of the framework is time-varying in order to pass through narrow passages. In this example, the trajectories of leaders are predefined (but known only to leaders). In practical missions, the leaders can generate trajectories in realtime according to task requirements and environment constraints. The formation tracking errors in Fig. 5.8(b) also converge to zero asymptotically, but with various convergence rates. Robots v_3 and v_4 have lower hierarchy and therefore will converge faster than the others. The simulation results in Fig. 5.8 validate Theorem 1 of our proposed approach.

5.6.2 Example scenario 2: Formation merging

Consider two robot swarms (Swarm A and Swarm B), with graphs constructed by Algorithm 1, that need to merge. Following Algorithm 2, three directed edges need to be added for the two frameworks to be merged, such that the two leaders of Swarm B become followers of Swarm A. Then, having been given a new target formation with desired bearing vectors such that the post-merged framework is IBR and BP,

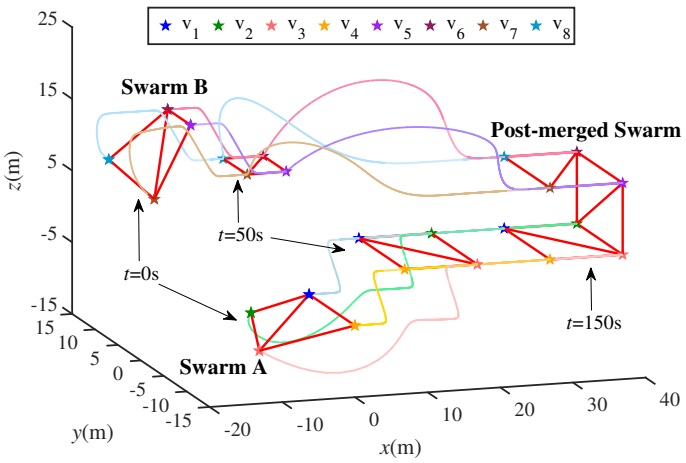


Figure 5.9: Formation evolution in Example scenario 2: Formation merging. (Reprinted from (Zhang et al., 2023).)

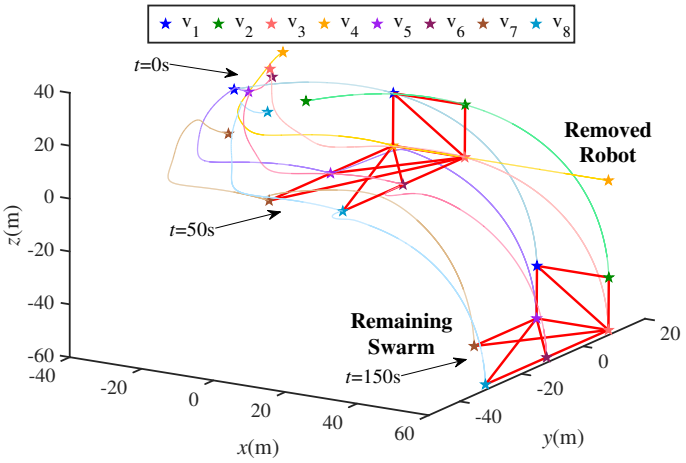


Figure 5.10: Formation evolution in Example scenario 3: Robot departure from formation. (Reprinted from (Zhang et al., 2023).)

the robots of the post-merged swarm will move under the formation control law to achieve the target formation.

A simulation example is shown in Fig. 5.9. In the first 50 s, the two frameworks each achieve the target square via control law (5.10), but with different scales. From 50 s, following Algorithm 2, three edges are newly added to merge the two frameworks, and the target cube is achieved at 150 s. Note that the scale of Swarm B is increased to that of Swarm A after performing the merging operation, which also demonstrates that Swarm B integrates successfully into Swarm A. The simulation results in Fig. 5.9 validate Theorem 2 of our proposed approach.

5.6.3 Example scenario 3: Robot departure from formation

In this example, the robot departure shown in Fig. 5.5 will be validated. In other words, robot v_4 will be removed and then Algorithm 3 will be performed to guarantee both the hierarchy and rigidity of the framework. Fig. 5.10 depicts the simulation results. From 0-50 s, the target formation will be achieved via the control law 5.10. From 50 s, robot v_4 will keep moving along a straight line and then robot v_5 will replace the position of v_4 in the framework, while v_7 will further replace the position of v_5, since v_5 is not an outer node robot. The simulation results show that the rigidity of the resulting framework is preserved, because the formation is not destroyed after the removal of robot v_4, thus validating Theorem 3 and Corollary 1 of our proposed approach.

5.6.4 Example scenario 4: Formation splitting

This example validates the splitting process as presented in Fig. 5.7(b), where a framework $(\mathcal{G}, p)$ (i.e., Pre-split Swarm in Fig. 5.11) including eight robots is split into two sub-frameworks $(\mathcal{G}_a, p_a)$ (i.e., Swarm A in Fig. 5.11) and $(\mathcal{G}_b, p_b)$ (i.e., Swarm B in Fig. 5.11), each with four robots.

The simulation result is given in Fig. 5.11. At $t = 50$ s, Z-link is constructed. From 50 s to 100 s, it can be noticed that the formation of the framework is maintained after Z-link construction, and also that the Z-link construction does not affect the rigidity

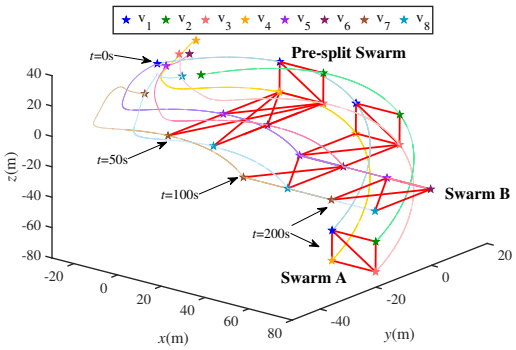


Figure 5.11: Formation evolution in Example scenario 4: Formation splitting. (Reprinted from (Zhang et al., 2023).)

of the framework. At $t = 100$ s, Z-link will be removed and two sub-frameworks will result. Thus, from 100 s to 200 s, robots v_5 and v_6 will be the leaders of Swarm B and the two sub-frameworks will move separately. The simulation results show that two sub-frameworks satisfy the bearing rigidity, because the formations are maintained after splitting, thus validating Theorem 4 and Lemma 7 of our approach.

5.7 Chapter conclusions

In this chapter, we demonstrated the construction of self-reconfigurable hierarchical frameworks for formation control of robot swarms, based on bearing rigidity under directed topologies.

Self-organized hierarchical control had previously been identified as a promising approach to ease the design and management of collective behaviors in robot swarms (Dorigo et al., 2020), and hierarchical frameworks had already been demonstrated in practical studies using the Mergeable Nervous Systems paradigm (Mathews et al., 2017; Zhu et al., 2020). However, strong theoretical foundations were still needed, especially for self-organized hierarchy to be viable for large-scale swarms of fast robots. In this chapter, we provided the first systematic and mathematically analyzable protocol for the implementation of self-reconfigurable hierarchical frameworks in robot swarms.

To enable self-organized hierarchical control with mathematically provable prop-

erties, we introduced a hierarchy property into conventional Henneberg construction and extended bearing rigidity to directed graphs with lower triangular structure. We studied self-reconfigurable hierarchical frameworks in three key reconstruction problems: merging, robot departure, and splitting. Finally, we demonstrated our frameworks by combining them with an example formation controller, and validated our theoretical results concerning hierarchy and rigidity preservation during reconfiguration via simulation experiments in four example scenarios.

Chapter 6

Proactive Fault Tolerance in Self-organized Hierarchy

This chapter[1] *addresses the problem of intermittent faults—transient errors that sporadically appear and disappear—in robot swarms. These faults pose significant obstacles to reliable communication and formation control, which are crucial for effective swarm coordination. Existing fault-tolerance strategies often focus on permanent faults and assume perfect communication, which is unrealistic in real-world scenarios. Building upon the self-organized hierarchical frameworks introduced in the previous chapters, we propose a novel proactive-reactive fault-tolerance method to manage intermittent faults. Proactively, robots employ an adaptive biased minimum consensus protocol to self-organize dynamic backup communication paths before faults occur. Then, they use a one-shot likelihood ratio test to detect faults early and reactively reroute communication through backup paths, maintaining reliable information flow. We demonstrate the effectiveness of this approach in multi-robot formations using self-organized hierarchical networks, showing that it prevents intermittent faults from disrupting convergence to desired formations.*

[1]The content of this chapter has been submitted to *IEEE Transactions on Robotics* for publication and is currently under review. Preprint available at arXiv:2509.19246.

6.1 Approach and contributions

In fault tolerance for multi-robot systems, both proactive and reactive mechanisms are important (Ghedini et al., 2017). In this chapter, we propose a novel *proactive–reactive* method to detect and mitigate IFs in robot swarms. In the proposed *proactive–reactive* method, the robots first use distributed consensus to preemptively self-organize dynamic backup communication paths before IFs are detected. Then, the robots compare information received via primary and backup paths to detect IFs, using a one-shot likelihood ratio test. When IFs are detected, the robots react by rerouting communication through the dynamic backup paths. In this chapter, we apply the proposed *proactive–reactive* method to a scenario of intermittently faulty relative positional information within multi-robot formations that have a hierarchical structure towards a fault-free leader, and demonstrate that the method mitigates IFs and robots are able to continue with the desired formations.

The main technical contributions of this chapter can be summarized as follows:

1. We address a current gap in robot swarm networking, specifically how to establish back-up communication paths for leader–follower formation control in a self-organized robot swarm. We address this gap by extending the biased minimum consensus (BMC) (Zhang and Li, 2017) protocol for shortest path planning in static graphs. We introduce the *adaptive biased minimum consensus* (ABMC) protocol for dynamic graphs—addressing time-varying topologies, node neighborhoods, and costs. We demonstrate that our ABMC protocol addresses the minimum-cost path problem, with two objectives integrated into a single cost function: to minimize the number of hops to the destination (the leader robot) and to minimize the degree of network congestion (by minimizing the occurrence of parallel edges). We provide the mathematical properties and stability analysis of the ABMC protocol as a distributed consensus mechanism in dynamic graphs with piecewise constancy, including providing the necessary and sufficient conditions to uniquely determine an equilibrium point representing a minimum-cost backup path.

2. We address a current gap in robot swarm fault tolerance, specifically tolerance against intermittent faults (IFs). We address this gap by proposing a novel *proactive–reactive* fault-tolerance strategy for detection and mitigation of IFs in robot swarms. Our proposed strategy uses the ABMC protocol to construct backup network layers and combines it with a distributed likelihood ratio (LR) protocol to dynamically reroute traffic in the constructed multiplex network. We propose the mathematical conditions and design the distributed algorithms for backup layer construction and for execution of the *proactive–reactive* strategy for IF detection and mitigation. We also provide the time and space complexity and efficiency properties of both distributed algorithms. Finally, we demonstrate the *proactive–reactive* fault-tolerance strategy in formations of 20 robots with moving leaders.

Regarding the existing literature on network routing (Bekmezci et al., 2013; Guillen-Perez et al., 2021; Oubbati et al., 2019), the proposed framework realizes a distributed, hierarchy-based multi-path routing protocol that combines proactive backup-path construction with reactive, detection-driven rerouting in response to intermittent data faults, tailored to self-organized hierarchical robot swarms.

6.2 Preliminaries

In this section, we introduce notations and graph-theoretic definitions used in this chapter, as well as the biased minimum consensus (BMC) protocol for distributed path planning in static undirected networks (Zhang and Li, 2017).

6.2.1 Directed graphs and hierarchical frameworks

Notation: Consider a swarm of $n \geq 2$ robots operating in the d-dimensional Euclidean space $\mathbb{R}^d$ (with $d = 2$ or 3). The robots are capable of establishing directed logical connections. The resulting graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a rooted directed graph where the vertex set $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ represents the robots and the edge set $\mathcal{E} = \{e_{ij} = (v_i,\ v_j) \mid v_i, v_j \in \mathcal{V},\ v_i \neq v_j\}$ models the logical connections between

them, where an edge e_{ij} indicates that the child v_i can receive information from the parent v_j.

Because the graph $\mathcal{G}$ is HHC-constructed (see Chapter 5), it is constructed incrementally from a two-robot kernel. The two starting robots are designated as the *leader* v_1 (the root) and the *first follower* v_2 (a unique child of the leader). This designation is not preassigned; any two robots can assume v_1 and v_2. Every other robot added to the graph is a *follower* with exactly two parents. For each robot v_i in $\mathcal{G}$, we define the following:

- parent set $\mathcal{P}_i$,
- neighbor set $\mathcal{N}_i$ (i.e., both the parents and children),
- in-degree δ_i^{in},
- out-degree δ_i^{out}, and
- *hierarchy* level $\mathcal{H}_i$, which is calculated as the hop count of the longest directed path from v_i to v_1.

The graph $\mathcal{G}$ is paired with the position configuration $\mathbf{Q}$, which describes the physical formation of the robots using pairwise relative positions. The configuration at time t is given by $\mathbf{Q}(t)$. For each robot pair (v_i, v_j) in the graph, we denote the true relative position of robot v_i with respect to v_j by $\mathbf{q}_{ij}$. The true relative position $\mathbf{q}_{ij}$ is supposed to be transmitted from robot v_j to v_i, however, the information might be corrupted by, e.g., sensor faults, communication errors, or malicious interference. Therefore, the (potentially compromised) relative position that is actually received by robot v_i is denoted as $\tilde{\mathbf{q}}_{ij}$.

6.2.2 The biased minimum consensus (BMC) protocol

The biased minimum consensus (BMC) protocol (Zhang and Li, 2017) is a distributed mechanism designed for static undirected graphs in which nodes update local states through interactions with neighbors, for the purpose of constructing paths to destination nodes. Each node v_i maintains a scalar state $s_i(t) \in \mathbb{R}$: its current estimate

of the quantity of interest (such as Euclidean distance). By relying only on information exchanged with neighbors (i.e., nodes directly connected to v_i by an edge in graph G), consensus on a path can be reached without requiring centralized control or monitoring.

The BMC protocol operates by first partitioning the node set $\mathcal{V}$ into two: $\mathcal{V}_1$, a set of destination nodes for paths, and the remainder set $\mathcal{V}_2 = \mathcal{V} \setminus \mathcal{V}_1$. All nodes v_i in $\mathcal{V}_2$ iteratively update their scalar state $s_i(t)$ to seek the minimum cost available through their respective neighbor sets $\mathcal{N}_i$ and thus collectively construct minimum-cost paths to destinations in $\mathcal{V}_1$. Each node v_i assesses costs according to weights $a_{ij} = a_{ji} > 0$ biasing its edges e_{ij}. The dynamics of this process are governed by:

$$\eta \dot{s}_i(t) = \begin{cases} 0, & v_i \in \mathcal{V}_1 \\ -s_i(t) + \min\limits_{v_j \in \mathcal{N}_i} \{s_j(t) + a_{ij}\}, & v_i \in \mathcal{V}_2 \end{cases} \tag{6.1}$$

where parameter $\eta > 0$ is a rate factor that influences the speed of convergence.

The states $s_i(t)$ gradually converge to a shared steady-state value s_i^*. In this equilibrium state, destination nodes in $\mathcal{V}_1$ retain their initial states, while the remaining nodes in $\mathcal{V}_2$ settle on the minimum biased state value among their neighbors, as follows (Zhang and Li, 2017):

$$s_i^*(t) = \begin{cases} s_i(0), & v_i \in \mathcal{V}_1 \\ \min\limits_{v_j \in \mathcal{N}_i} \{s_j(t) + a_{ij}\}, & v_i \in \mathcal{V}_2 \end{cases} \tag{6.2}$$

When applying the BMC protocol to the shortest path problem based on Euclidean distance (Mo et al., 2021; Zhang and Li, 2017), a node's state $s_i(t)$ can be interpreted as the distance from node v_i to a destination, and the bias term a_{ij} as the distance from v_i to v_j. Through iterative updates, guided by Bellman's optimality principle, the protocol can establish shortest-distance paths from any source node to the given destination node(s) (Zhang and Li, 2017).

6.3 Problem Statement

In a self-organizing hierarchical swarm (see Chapter 4), a robot occupying the leader position uses information accumulated from its downstream robots to make decisions and then issues instructions to its downstream robots. However, the leader interacts directly only with its direct children, using multi-hop communication to interact with the rest of the robots in the swarm. Therefore, maintaining reliable multi-hop communication paths between the leader and all other robots in the swarm is crucial for effective coordination. These communication paths can become disrupted or inefficient when intermittent faults (IFs) are present. The objective of this chapter is to develop a proactive-reactive fault tolerance mechanism to mitigate the effect of IFs on a swarm's ability to maintain accurate positional information for performing formation tasks, in a robot swarm with a self-organizing hierarchical architecture (HHC-constructed). The following three questions will be addressed:

1. Given a swarm of n robots in an HHC-constructed formation, how can the robots collectively self-organize dynamic minimum-cost backup paths to the leader that maintain the hierarchy conditions of the original graph and also adapt to its reconfigurations, using only local information from nearby robots?

2. Given the constructed backup paths, how can the robots use them to detect the presence of IFs in their original paths to the leader?

3. Given some detected IFs, how can the robots use the constructed backup paths to mitigate the effect of those IFs while present, and to switch back to their original paths to the leader once the respective IFs have stabilized?

6.4 Adaptive Minimum-cost Backup Paths

In a scenario in which some robots in a swarm are subject to IFs, a robot v_i can circumvent faulty information being transmitted by an intermediary robot (i.e., one lying between it and the leader v_1) by constructing a new “backup" path to v_1 that circumvents the faulty robot. This section presents our distributed method to

construct backup paths that adapt to dynamic networks, by allowing follower robots to independently determine their own upstream connections.

Using locally available information, v_i selects a robot from among those in its communication range to become its preferred backup parent (that is, its backup next hop towards the leader v_1). As follower robots in a swarm repeatedly update their preferred backup parents at each step, the resulting chains of distributed parent choices form a *minimum-cost* backup path $\mathcal{B}_i = \{v_i, \ldots, v_1\}$ for each follower robot v_i.

6.4.1 Adaptive biased minimum consensus protocol (ABMC)

To address the minimum-cost path problem in rooted directed graphs, we propose our adaptive biased minimum consensus (ABMC) protocol. Because the scenario we consider is that of communication paths among networked robots, we aim to construct paths that are both 1) efficient (minimizing the number of hops to the root) and 2) maximally disjoint (minimizing communication congestion and bottlenecks from different paths sharing common edges and vertices), using a single composite cost. For this aim, our ABMC protocol leverages the structure of the graph by restricting next-hop candidates to upstream nodes and by considering the outdegree δ^{out} of the robots.

The ABMC protocol extends the BMC protocol by introducing a dynamic neighbor set and dynamic bias term. The original BMC was designed for selecting pre-existing edges from a static network based on static biases. The ABMC, by contrast, is designed to construct new edges that might not be present in the original network, based on the dynamic topology of the original network, the dynamic positions of the robots, and the dynamic biases associated to potential new paths. The ABMC is designed as follows:

$$\eta \dot{s}_i(t) = \begin{cases} 0, & v_i \in \mathcal{V}_1 \\ -s_i(t) + \min\limits_{v_j \in \mathcal{P}_i^{\text{cand}}(t)} \left\{ s_j(t) + a_{ij}(t) \right\}, & v_i \in \mathcal{V}_2 \end{cases} \tag{6.3}$$

where, $s_i(t)$ is the state value representing the estimated number of hops at time t from robot v_i to the leader, $\mathcal{P}_i^{\text{cand}}(t)$ is the candidate parent set of robot v_i at time t, $a_{ij}(t)$ is the bias against selecting robot v_j as the next hop for communication at time t, and $a_{ij} \neq a_{ji}$. The parameter η is the convergence rate factor, determining how quickly the hop count estimates converge to the final values. $\mathcal{V}_1$ is the set of the leader and first follower, for which the hop count estimate remains constant, and $\mathcal{V}_2$ is the set of all other robots in the network.

The candidate parent set $\mathcal{P}_i^{\text{cand}}(t)$ in Eq. (6.3) is a departure from the neighbor set N_i in the existing BMC (Eq. (6.1)), because it is dynamic, includes nodes that are not connected to v_i in the original graph, and leverages graph directionality towards the destination. The candidate parent set $\mathcal{P}_i^{\text{cand}}(t)$ includes any node v_j that meets the following criteria at time t:

1. **In-range:** v_j is within communication range r of v_i.
2. **Non-adjacent:** v_i and v_j are not connected by an edge in the primary network G).
3. **Leader-proximate:** v_j is fewer hops than v_i from the leader v_1 (i.e., $\mathcal{H}_j < \mathcal{H}_i$).

Formally, these criteria are given by:

$$\mathcal{P}_i^{\text{cand}}(t) = \left\{ v_j \in \mathcal{V} \setminus \{v_i\} \;\middle|\; (v_i, v_j) \notin \mathcal{E},\ \mathcal{H}_j < \mathcal{H}_i, |v_i, v_j| \leq r \right\} \tag{6.4}$$

where $|i, j|$ denotes the Euclidean distance between i and j, and $r > 0$ is the communication range.

In BMC, the bias term is static and typically represents Euclidean distances between fixed positions associated with nodes of the original (static) network. By contrast, in ABMC, the bias term is dynamic and accounts for both hierarchy differences and the potential network congestion at each node. The dynamic bias $a_{ij}(t)$ in Eq. (6.3) is defined as follows:

$$a_{ij}(t) = \max\left\{ 1 - \rho\Big(\mathcal{H}_i(t) - \mathcal{H}_j(t)\Big) \ + \psi\, \phi\Big(\delta_j^{\text{out}}(t),\, \kappa_d\Big), \gamma \right\} \tag{6.5}$$

where $\rho > 0$ is a weighting factor adjusting the influence of hierarchy differences on the bias term, and $\psi \geq 0$ serves as a penalty weight that scales the impact of node congestion on the dynamic bias. We define the congestion penalty function as $\phi\Big(\delta_j^{\text{out}}(t),\ \kappa_d\Big) = \max\Big\{0,\ \delta_j^{\text{out}}(t) - \kappa_d\Big\}$ where $\delta_j^{\text{out}}(t)$ is the outdegree of robot v_j at time t, κ_d is the outdegree threshold, and $\gamma > 0$ is a small positive value ensuring $a_{ij}(t)$ never falls below a specified minimum cost.

Remark 1: The parameter ρ will usually be close to 1, ensuring moderate hierarchy differences do not excessively lower the term $1 - \rho(\mathcal{H}_i(t) - \mathcal{H}_j(t))$. This prevents the bias term from frequently saturating at its lower bound, allowing it to remain responsive to meaningful hierarchical variations without overly penalizing larger hierarchy gaps.

Remark 2: The parameter ψ is used to design how strongly the penalty term discourages node usage once the outdegree $\delta_j^{\text{out}}(t)$ surpasses a threshold κ_d. A sufficiently large value of ψ can simulate a hard constraint, virtually eliminating paths through overloaded nodes, or a moderate ψ can permit a balance between hop minimization and congestion management. Simultaneously, a higher κ_d value increases the number of next-hop candidates, but also increases the likelihood of bottlenecks, while a lower κ_d reduces the search space, thus lowering the likelihood of bottlenecks but also reducing the availability of next-hop candidates.

Remark 3: The relationship between hierarchy difference and hop count is non-monotonic because the topology of the primary network $\mathcal{G}$ determines the hierarchy levels, while the position configuration $\mathbf{Q}$ determines which robots are in range r for robot v_i and therefore are candidates to be in the set $\mathcal{P}_i^{\text{cand}}$. In short, larger hierarchy differences do not necessarily correspond to fewer hops in the backup path.

Hierarchy levels depend on the dynamic reconfiguration of the primary network $\mathcal{G}$. In practice, a stable topology is often maintained over extended periods, allowing us to model the hierarchy levels as piecewise constant functions over time intervals between reconfiguration events. Let $\{t_k\}_{k=0}^{N} \subset [0, T]$ denote the discrete time instants at which reconfiguration events occur, where N is the total number of events. These instants satisfy $0 = t_0 < t_1 < \cdots < t_N \leq T$, with T representing the total operational

time. Between these reconfiguration events, the hierarchy levels remain constant:

$$\mathcal{H}_i(t) = \mathcal{H}_i(t_k), \quad \forall t \in [t_k, t_{k+1}), \quad \forall v_i \in \mathcal{V} \tag{6.6}$$

This implies that the hierarchy difference $\mathcal{H}_i(t) - \mathcal{H}_j(t)$ and, consequently, the bias term $a_{ji}(t)$ defined in Eq. (6.5), remain constant within each interval $[t_k, t_{k+1})$:

$$a_{ij}(t) = a_{ij}(t_k), \quad \forall t \in [t_k, t_{k+1}), \quad \forall v_i, v_j \in \mathcal{V} \tag{6.7}$$

When a reconfiguration of $\mathcal{G}$ occurs, hierarchy levels are recalculated as:

$$\mathcal{H}_i(t_k^+) = \max_{v_j \in \mathcal{P}_i} \{\mathcal{H}_j(t_k^+)\} + 1, \quad \forall v_i \in \mathcal{V} \tag{6.8}$$

where t_k^+ denotes the time immediately after t_k.

Remark 4: We distinguish between two different categories of topology changes in our scenario: transient disruptions versus desired semi-permanent changes. On one hand, there can be transient interruptions of some edges, for example because of temporary sensor occlusion or communication disruption. These transient topology interruptions because of minor edge disturbances are negligible to the ABMC consensus time scale, because such disturbances are much shorter-lived than the ABMC convergence process. On the other hand, there can be semi-permanent reconfigurations of the desired topology, which trigger HHC-reconstruction events (see (Zhang et al., 2023)) to rebuild the graph and the hierarchy states. After such a reconfiguration event, the ABMC protocol then adapts to the resulting semi-permanent graph, re-converging on new backup paths (until the next reconfiguration event). Accordingly, between two reconfiguration instants t_k and t_{k+1}, both the hierarchy states and the ABMC candidate parent sets are treated as piecewise constant; the topology is fixed within each interval $[t_k, t_{k+1})$. This permits the use of standard convergence analysis for consensus on each interval. The overall stability of the protocol is preserved provided the dwell time $t_{k+1} - t_k$ is sufficiently large relative to the convergence rate set by η (Moreau, 2005; Olfati-Saber and Murray, 2004; Ren and Beard, 2005).

The operation of the ABMC protocol on an example HHC-constructed primary network $\mathcal{G}$ and formation $\mathbf{Q}$ is illustrated in Fig. 6.1. The figure shows how a robot identifies potential next hops within its communication range r and creates a backup edge towards the leader.

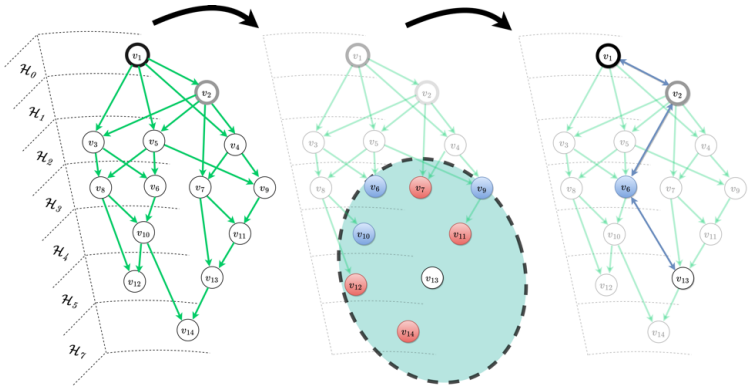


Figure 6.1: Illustration of constructing a minimum-cost backup path using the ABMC protocol, in an example hierarchical swarm of 14 robots with seven hierarchy levels $\mathcal{H}_0$ to $\mathcal{H}_6$. The left panel shows the primary network $\mathcal{G}$. The middle panel shows how robot v_{13} checks the robots in its communication range (dashed circle) and determines its potential next hops (blue) toward the leader v_1, by excluding direct parents (v_7 and v_{11}; red) and any robots that fail to meet hierarchy requirement (v_{12} and v_{14}; also red). The right panel depicts the minimum-cost backup path (blue edges) from v_{13} to v_1 that results from each robot in the swarm executing several iterations of the ABMC protocol, in a fully decentralized way.

6.4.2 Mathematical properties and stability analysis of ABMC protocol

Standard consensus analysis using graph Laplacians relies on symmetric interactions (Bullo et al., 2018). The ABMC introduces directional bias terms, breaking symmetry and invalidating spectral methods to understand convergence behavior (i.e., methods that rely on the eigenvalues and eigenvectors of the graph Laplacian). We therefore provide specialized mathematical foundations and stability analysis for the asymmetric interactions of the ABMC protocol.

Let $\beta_i(t)$ represent the right-hand side of Eq. (6.3). The upper bound of β_i,

denoted by $\bar{\beta}$, is defined as the maximum value of β_i across all robots, given by $\bar{\beta} = \max_{v_i \in \mathcal{V}} \{\beta_i\}$. The set of robots that attain this maximum is denoted by $\overline{\mathcal{S}} = \arg\max_{v_i \in \mathcal{V}} \{\beta_i\}$. Similarly, the lower bound $\underline{\beta}$ and the corresponding set $\underline{\mathcal{S}}$ are defined analogously as $\underline{\beta} = \min_{v_i \in \mathcal{V}} \{\beta_i\}$ and $\underline{\mathcal{S}} = \arg\min_{v_i \in \mathcal{V}} \{\beta_i\}$, respectively.

Let $\mathcal{P}_i^{\text{back}}$ represent the backup parents that v_i selects from among its candidate parents $\mathcal{P}_i^{\text{cand}}$. $\mathcal{P}_i^{\text{back}}$ is defined as a subset of $\mathcal{P}_i^{\text{cand}}$ that minimizes the sum of the state value and the bias term, as follows:

$$\mathcal{P}_i^{\text{back}} := \arg \min_{v_j \in \mathcal{P}_i^{\text{cand}}(t)} \{s_j(t) + a_{ij}(t)\} \tag{6.9}$$

Note that, because the leader robot v_1 originates the positional information in the swarm, its backup parent set is empty, $\mathcal{P}_1^{\text{back}} = \emptyset$: it does not require an alternative information path.

We now present a series of four lemmas that establish the properties of the ABMC protocol. The first lemma demonstrates that the robots' hop count estimates evolve in a controlled manner. Specifically, it establish that the maximum and minimum values of $\beta_i(t)$—which are directly related to the rates of change of the robots' hop count estimates—are monotonically non-increasing and non-decreasing, respectively. This ensures that the updates neither accelerate nor decelerate in an unbounded manner.

Lemma 1: The upper bound $\bar{\beta}(t) = \max_{v_i \in \mathcal{V}} \beta_i(t)$ is monotonically non-increasing, and the lower bound $\underline{\beta}(t) = \min_{v_i \in \mathcal{V}} \beta_i(t)$ is monotonically non-decreasing.

Proof: Recall $\beta_i(t)$ and the definition of $\mathcal{P}_i^{\text{back}}$ in Eq. (6.9). For $v_i \in \mathcal{V}_1$, $\dot{\beta}_i(t) = 0$. For $v_i \in \mathcal{V}_2$, set $m_i(t) = \min_{v_j \in \mathcal{P}_i^{\text{cand}}(t)} \Big\{ s_j(t) + a_{ij}(t) \Big\}$. Because the min operator may be non-differentiable, we use the upper right Dini derivative D^+ and Clarke's chain rule (Clarke, 1990). It yields $D^+ m_i(t) \in \text{conv} \Big\{ \dot{s}_j(t) + \dot{a}_{ij}(t) : v_j \in \mathcal{P}_i^{\text{back}} \Big\}$. By Remark 4, $a_{ij}(t)$ is piecewise constant on $[t_k, t_{k+1})$, hence $\dot{a}_{ij}(t) = 0$ there. Therefore $D^+ \beta_i(t) \in -\dot{s}_i(t) + \text{conv} \Big\{ \dot{s}_j(t) : v_j \in \mathcal{P}_i^{\text{back}} \Big\}$.

Thus, there exist weights $w_j \geq 0$ with $\sum_{v_j \in \mathcal{P}_i^{\text{back}}} w_j = 1$ such that

$$D^+\beta_i(t) \;=\; -\,\dot{s}_i(t) \;+\; \sum_{v_j \in \mathcal{P}_i^{\text{back}}} w_j\,\dot{s}_j(t) \tag{6.10}$$

Using the update rule $\eta\,\dot{s}_\ell(t) = \beta_\ell(t)$, we have $\dot{s}_\ell(t) = \beta_\ell(t)/\eta$ for all ℓ. Substituting into (6.10) gives

$$D^+\beta_i(t) \;=\; \frac{1}{\eta} \sum_{v_j \in \mathcal{P}_i^{\text{back}}} w_j\Big(\beta_j(t) - \beta_i(t)\Big) \tag{6.11}$$

Now consider $\bar{\beta}(t) = \max_{v_i \in \mathcal{V}} \beta_i(t)$. By Clarke's max rule,

$$D^+\bar{\beta}(t) \;\leq\; \max_{v_i \in \overline{\mathcal{S}}(t)} D^+\beta_i(t) \;\leq\; \sum_{v_i \in \overline{\mathcal{S}}(t)} \delta_i\, D^+\beta_i(t) \tag{6.12}$$

for some $\delta_i \geq 0$ with $\sum_{v_i \in \overline{\mathcal{S}}(t)} \delta_i = 1$, where $\overline{\mathcal{S}}(t) = \{v_i : \beta_i(t) = \bar{\beta}(t)\}$. Substituting (6.11) yields

$$D^+\bar{\beta}(t) \;\leq\; \sum_{v_i \in \overline{\mathcal{S}}(t)} \sum_{v_j \in \mathcal{P}_i^{\text{back}}} \frac{\delta_i w_j}{\eta} \Big(\beta_j(t) - \beta_i(t)\Big) \tag{6.13}$$

For $v_i \in \overline{\mathcal{S}}(t)$ we have $\beta_i(t) = \bar{\beta}(t) \geq \beta_j(t)$ for all v_j, hence each difference $\beta_j(t) - \beta_i(t) \leq 0$. Since $\delta_i \geq 0$, $w_j \geq 0$, and $\eta > 0$, every term in (6.13) is ≤ 0, so $D^+\bar{\beta}(t) \leq 0$. Thus $\bar{\beta}(t)$ is monotonically non-increasing.

For the lower envelope $\underline{\beta}(t) = \min_{v_i} \beta_i(t)$, note that $\underline{\beta}(t) = -\max_{v_i}(-\beta_i(t))$. Applying the same argument to $-\beta_i(t)$ gives $D^+\underline{\beta}(t) \geq 0$, i.e., $\underline{\beta}(t)$ is monotonically non-decreasing. This completes the proof. ■

The second lemma describes the long-term interaction dynamics among the robots. It shows that, over time, the robots influencing the state of those achieving the maximum and minimum rate of change are themselves among the robots achieving the maximum and minimum rate of change, respectively.

Lemma 2: As $t \to +\infty$, for every robot v_i that attains the upper bound $\bar{\beta}(t)$ (i.e., $v_i \in \overline{\mathcal{S}}$), its backup parent set satisfies $\mathcal{P}_i^{\text{back}} \subset \overline{\mathcal{S}}$. Similarly, for every robot v_i that attains the lower bound $\underline{\beta}(t)$ (i.e., $v_i \in \underline{\mathcal{S}}$), we have $\mathcal{P}_i^{\text{back}} \subset \underline{\mathcal{S}}$.

Proof: From Lemma 1, we have $\underline{\beta}(0) \leq \underline{\beta}(t) \leq \beta_i(t) \leq \bar{\beta}(t) \leq \bar{\beta}(0),\ \forall\, v_i \in$

$\mathcal{V},\ t \geq 0$. Since $\bar{\beta}(t)$ is monotonically non-increasing and bounded below, the Monotone Convergence Theorem (Royden and Fitzpatrick, 1988) implies $\lim_{t\to+\infty} \bar{\beta}(t) = c_1,$ with $\underline{\beta}(0) \leq c_1 \leq \bar{\beta}(0)$. By definition, for every v_i in $\overline{\mathcal{S}} = \{v_i \in \mathcal{V} \mid \beta_i(t) = \bar{\beta}(t)\}$, we have $\lim_{t\to+\infty} \beta_i(t) = c_1$. For any $v_j \in \mathcal{P}_i^{back}$, noting that $\beta_j(t) \leq \bar{\beta}(t)$, we obtain $\lim_{t\to+\infty} \beta_j(t) \leq c_1$. As $t \to +\infty$ (with $\bar{\beta}(t) \to c_1$), using the upper right Dini derivative and Eq. (6.13) gives

$$0 \ \geq \ \limsup_{t\to+\infty} D^+\bar{\beta}(t) \ \geq \ \limsup_{t\to+\infty} \sum_{v_i\in\overline{\mathcal{S}}\cap\mathcal{V}_2} \sum_{v_j\in\mathcal{P}_i^{back}} \frac{\delta_i w_j}{\eta} \Big(\beta_j(t) - c_1\Big) \tag{6.14}$$

where $\delta_i \geq 0$, $\sum_{v_i\in\overline{\mathcal{S}}} \delta_i = 1$, and $w_j \geq 0$, $\sum_{v_j\in\mathcal{P}_i^{back}} w_j = 1$. Since each coefficient is nonnegative, the only way for the limit superior of this weighted sum of nonpositive terms to be zero is that each active term satisfies $\beta_j(t) - c_1 \to 0$. Hence

$$\lim_{t\to+\infty} \beta_j(t) = c_1, \quad \forall\, v_j \in \mathcal{P}_i^{back} \tag{6.15}$$

By LaSalle's Invariance Principle (Isidori, 2013), the system trajectories converge to the largest invariant set where $D^+\bar{\beta}(t) = 0$. In this invariant set, we must have $\mathcal{P}_i^{back} \subset \overline{\mathcal{S}}, \quad \forall\, v_i \in \overline{\mathcal{S}}$, i.e., as $t \to +\infty$, robots with maximum $\beta_i(t)$ interact only with robots that also achieve $\beta_i(t) = c_1$. This completes the proof. ■

The third lemma shows the boundedness of the protocol states. This ensures that the state values of the robots do not increase indefinitely and thus protocol stability is maintained.

Lemma 3: $s_i(t) \leq S_{\max},\ t > 0,\ \forall v_i \in \mathcal{V}_2$.

Proof: Since the candidate parent set $\mathcal{P}_i^{\text{cand}}(t)$ is finite, there exists at least one robot $v_n \in \mathcal{P}_i^{back}$ such that $\min_{v_j\in\mathcal{P}_i^{\text{cand}}(t)}\{s_j(t) + a_{ij}(t)\} = s_n(t) + a_{in}(t)$. Therefore, by the update rule we have $s_i(t) = s_n(t) + a_{in}(t) - \beta_i(t)$. Since $\beta_i(t) \geq \underline{\beta}(0)$, we have

$$s_i(t) \leq s_n(t) + a_{in}(t) - \underline{\beta}(0) \tag{6.16}$$

More generally, for any $v_j \in \mathcal{P}_i^{\text{cand}}(t)$,

$$s_i(t) = \min_{v_\ell \in \mathcal{P}_i^{\text{cand}}(t)} \{s_\ell(t) + a_{i\ell}(t)\} - \beta_i(t) \;\leq\; s_j(t) + a_{ij}(t) - \underline{\beta}(0) \tag{6.17}$$

so (6.16) applies to each backup parent.

To extend this local bound to a global one, we use the *reachability via backup parents* assumption: for every robot $v_i \in \mathcal{V}_2$ there is a finite directed backup path connecting it to the leader robot $v_1 \in \mathcal{V}_1$, say $\{v_i, \ldots, v_n, \ldots, v_1\}$, where for each consecutive pair (v_n, v_{n+1}) along the path we have $v_{n+1} \in \mathcal{P}_n^{\text{cand}}(t)$. Then, for each link along the path, $s_n(t) \leq s_{n+1}(t) + a_{n,n+1}(t) - \underline{\beta}(0)$. If we define $C = a_{\max} - \underline{\beta}(0)$, where $a_{\max}$ is an upper bound on all link costs, then for each link we have $s_n(t) \leq s_{n+1}(t) + C$. By chaining these inequalities from the robot v_i to the leader robot v_1 (across, say, $n-1$ links) gives $s_i(t) \leq s_1(0) + (n-1)C$. Let $n_{\max}$ be the maximum number of hops needed for any robot in $\mathcal{V}_2$ to reach the leader robot along backup parents. Then, by taking the worst-case path we define $S_{\max} = s_1(0) + (n_{\max} - 1)C$. Hence, for every $v_i \in \mathcal{V}_2$ and all $t > 0$, $s_i(t) \leq S_{\max}$. This completes the proof. ■

Finally, the fourth lemma demonstrates that as time approaches infinity, the leader will be included in the set of robots with the minimum state value. By aligning its state with the minimum, the leader follows the same protocol as other robots, promoting network stability and preventing divergence.

Lemma 4: As $t \to +\infty$, $\underline{\mathcal{S}} \cap \mathcal{V}_1 \neq \emptyset$, where $\underline{\mathcal{S}}$.

Proof: Recall that $\beta_1(t) = 0$ for all t. Suppose, for contradiction, that $\underline{\mathcal{S}}(t) \cap \mathcal{V}_1 = \emptyset$ for arbitrarily large t. Then, since $\underline{\beta}(t) = \min_i \beta_i(t)$ and $\beta_1(t) = 0$, we must have $\underline{\beta}(t) < 0$ for those t.

From the definition of $\mathcal{P}_i^{back}$, we have $\mathcal{P}_i^{back} \neq \emptyset$ for every $v_i \in \underline{\mathcal{S}}(t)$. Moreover, by Lemma 2 (lower-bound case), $\mathcal{P}_i^{back} \subset \underline{\mathcal{S}}(t) \quad \text{for all} \quad v_i \in \underline{\mathcal{S}}(t)$ for sufficiently large t. For any such t, pick $v_i \in \underline{\mathcal{S}}(t)$ and any $v_n \in \mathcal{P}_i^{back}$. Using

$$\begin{aligned} \beta_i(t) \;&=\; -\,s_i(t) \;+\; \min_{v_j \in \mathcal{P}_i^{\text{cand}}(t)} \Big\{ s_j(t) + a_{ij}(t) \Big\} \\ &=\; -\,s_i(t) \;+\; s_n(t) + a_{in}(t) \end{aligned} \tag{6.18}$$

we obtain

$$s_i(t) \;=\; s_n(t) + a_{in}(t) - \beta_i(t) \tag{6.19}$$

Since $\beta_i(t) = \underline{\beta}(t) < 0$ and $a_{in}(t) \geq \gamma > 0$, (6.19) gives the strict inequality

$$s_i(t) \;>\; s_n(t) + \gamma \;>\; s_n(t) \tag{6.20}$$

for all $v_n \in \mathcal{P}_i^{back} \subset \underline{\mathcal{S}}(t)$.

Now define $s_m(t) := \min_{v_\ell \in \underline{\mathcal{S}}(t)} s_\ell(t)$ and choose $v_i \in \underline{\mathcal{S}}(t)$ attaining this minimum. Then $s_i(t) = s_m(t) \leq s_n(t)$ for every $v_n \in \underline{\mathcal{S}}(t)$, which contradicts (6.20). Therefore, our supposition must be false, and $\underline{\mathcal{S}}(t) \cap \mathcal{V}_1 \neq \emptyset$ for all sufficiently large t. This completes the proof. ■

Having established the properties of the ABMC protocol through lemmas 1–4, we now present two theorems that demonstrate its stability—that is, the ability of the consensus protocol to reliably reach a state of equilibrium (Ren et al., 2005).

Theorem 1: Fix $[t_k, t_{k+1})$ on which $\mathcal{H}$, $\mathcal{N}_i$, and a_{ij} are constant. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an HHC-constructed graph with the leader robot v_1 and first follower v_2; destination set $\mathcal{V}_1 = \{v_1, v_2\}$, remainder set $\mathcal{V}_2 = \mathcal{V} \setminus \mathcal{V}_1$. For each $v_i \in \mathcal{V}_2$, the candidate parent set $\mathcal{P}_i^{\text{cand}}$ is defined in Eq. (6.4) and *reachability* of $\mathcal{V}_1$ via the candidate parents holds: $\forall\, v_i \in \mathcal{V}_2\, \exists\, m$ and a sequence $v_{k_0}, \ldots, v_{k_m}$ with $v_{k_0} = v_i$, $v_{k_{\ell+1}} \in \mathcal{P}_{k_\ell}^{\text{cand}}$, and $v_{k_m} \in \mathcal{V}_1$. Each robot v_i maintains a scalar state $s_i(t) \in \mathbb{R}$, interpreted as its current estimate of the minimum cumulative bias from v_i to $\mathcal{V}_1$ along converged paths. Each robot adheres to the ABMC protocol given in Eq. (6.3). Then on $[t_k, t_{k+1})$, there exists a unique $s^* \in \mathbb{R}^{|\mathcal{V}|}$ for each $v_i \in \mathcal{V}$ with:

$$s_i^*(t) = \begin{cases} s_i(0), & v_i \in \mathcal{V}_1 \\ \min\limits_{v_j \in \mathcal{P}_i^{\text{cand}}(t)} \left\{ s_j^*(t) + a_{ij}(t) \right\}, & v_i \in \mathcal{V}_2 \end{cases} \tag{6.21}$$

For any $s(0) \in \mathbb{R}^{|\mathcal{V}|}$, the solution $s(t)$ converges globally and asymptotically to s^*.

Proof:

Building on Lemma 1, there exist constants $c_1, c_2 \in \mathbb{R}$ such that $\lim_{t\to\infty} \bar{\beta}(t) = c_1$

and $\lim_{t\to\infty} \underline{\beta}(t) = c_2$. By Lemma 4, there exists an increasing, unbounded sequence $\{t_k\}_{k\in\mathbb{N}}$ with $t_k \to \infty$ such that $\underline{\mathcal{S}}(t_k) \cap \mathcal{V}_1 \neq \emptyset$ for all $k \in \mathbb{N}$. For each k, pick $v_i \in \underline{\mathcal{S}}(t_k) \cap \mathcal{V}_1$. Since $\beta_j(t) = 0$ for all $v_j \in \mathcal{V}_1$ and all t, we have $\underline{\beta}(t_k) = 0$ for every k. Hence $\liminf_{t\to\infty} \underline{\beta}(t) = 0$. Because $\underline{\beta}(t)$ has a (finite) limit c_2, it follows that $c_2 = 0$; that is, $\lim_{t\to\infty} \underline{\beta}(t) = 0$.

Now, consider the definition of $\bar{\beta}(t)$, which implies $\bar{\beta}(t) \geq \underline{\beta}(t)$. For any time t and any robot $v_i \in \overline{\mathcal{S}}(t)$, the ABMC protocol dictates that $\eta \dot{s}_i(t) = \beta_i(t) = \bar{\beta}(t)$. Summing over all robots and using that every non-maximizer has $\beta_i(t) \geq \underline{\beta}(t)$, we obtain the pointwise bound

$$\eta \sum_{v_i\in\mathcal{V}} \dot{s}_i(t) = \sum_{v_i\in\mathcal{V}} \beta_i(t) \;\geq\; |\overline{\mathcal{S}}(t)|\,\bar{\beta}(t) \;+\; \Big(|\mathcal{V}| - |\overline{\mathcal{S}}(t)|\Big)\,\underline{\beta}(t) \tag{6.22}$$

Fix $\varepsilon > 0$. Since $\underline{\beta}(t) \to 0$, there exists T_ε such that $\underline{\beta}(t) \geq -\varepsilon$ for all $t \geq T_\varepsilon$. Integrating (6.22) from T_ε to τ and using Lemma 3 (boundedness of $s_i(t)$) yields that the left-hand side is uniformly bounded in τ, whereas the right-hand side is

$$\int_{T_\varepsilon}^{\tau} |\overline{\mathcal{S}}(t)|\,\bar{\beta}(t)\,dt \;-\; (|\mathcal{V}| - 1)\,\varepsilon\,(\tau - T_\varepsilon) \tag{6.23}$$

Since $|\overline{\mathcal{S}}(t)| \geq 1$, if $\limsup_{t\to\infty} \bar{\beta}(t) > 0$ the integral would diverge to $+\infty$ (for ε arbitrarily small), a contradiction. Hence $\lim_{t\to\infty} \bar{\beta}(t) = 0$. Combined with $\lim_{t\to\infty} \underline{\beta}(t) = 0$, we have $\max_{v_i\in\mathcal{V}} |\beta_i(t)| \to 0$, which implies $\dot{s}_i(t) \to 0$ for all $v_i \in \mathcal{V}$.

Let s^∞ be any limit point of $s(t)$. Passing to the limit in the ABMC update gives, for each $v_i \in \mathcal{V}$,

$$0 = \lim_{t\to\infty} \beta_i(t) = \begin{cases} 0, & v_i \in \mathcal{V}_1, \\ -\,s_i^\infty + \min_{v_j\in\mathcal{P}_i^{\text{cand}}} \{s_j^\infty + a_{ij}\}, & v_i \in \mathcal{V}_2 \end{cases}$$

so s^∞ satisfies Eq. (6.21). By uniqueness of the equilibrium, $s^\infty = s^*$ and therefore $s(t) \to s^*$ globally. This establishes global asymptotic stability of the equilibrium. ■

Building upon the global convergence of the ABMC protocol, we now examine

the relationship between the equilibrium state and the minimum-cost backup path. The following theorem establishes the equivalence between the equilibrium point of the protocol and the solution to the minimum-cost path problem.

Theorem 2: If the initial state of the leader robot is $s_1(0) = 0$, then the equilibrium point of the ABMC protocol serves as a solution to the minimum-cost variant of the shortest-path problem.

Proof: In the classical shortest-path problem, the goal is to determine a path that minimizes the cumulative (nonnegative) edge cost from any node v_i to a designated source, with the corresponding optimal cost s_i satisfying the recursive relation

$$s_i \;=\; \min_{v_j \in \mathcal{P}_i^{\text{cand}}} \{\, s_j + c_{ij} \,\}, \qquad s_{\text{source}} = 0 \tag{6.24}$$

where $c_{ij} \geq 0$ is the cost of (v_i, v_j). This is Bellman's equation (Bellman, 1958).

On the fixed interval $[t_k, t_{k+1})$, the sets $\mathcal{P}_i^{\text{cand}}$ and biases a_{ij} are constant. In our formulation, the state s_i represents the minimum cumulative bias along a directed backup path from v_i to the leader v_1, and the classical edge cost c_{ij} is replaced by the bias $a_{ij} \geq \gamma > 0$. Hence the recursion becomes $s_i \;=\; \min_{v_j \in \mathcal{P}_i^{\text{cand}}} \{\, s_j + a_{ij} \,\}, \qquad s_1 = 0$. This is exactly Bellman's equation with edge costs a_{ij}.

Define the (time-invariant on $[t_k, t_{k+1})$) Bellman operator

$$(\mathfrak{T}s)_i \;\coloneqq\; \min_{v_j \in \mathcal{P}_i^{\text{cand}}} \{\, s_j + a_{ij} \,\}, \quad i \in \mathcal{V}_2, \qquad (\mathfrak{T}s)_1 \coloneqq 0 \tag{6.25}$$

The operator $\mathfrak{T}$ is monotone and nonexpansive in the max norm: $\|\mathfrak{T}s - \mathfrak{T}\tilde{s}\|_\infty \leq \|s - \tilde{s}\|_\infty$. Moreover, by construction of $\mathcal{P}_i^{\text{cand}}$ with $\mathcal{H}_j < \mathcal{H}_i$, every admissible hop strictly decreases hierarchy, so directed backup paths are acyclic and finite, and v_1 is reachable from every $v_i \in \mathcal{V}_2$. On a finite acyclic graph with nonnegative edge costs and boundary $s_1 = 0$, the system $s = \mathfrak{T}s$ has a unique solution equal to the vector of minimum cumulative costs to v_1 (standard dynamic programming/shortest-path argument, e.g., induction over increasing hierarchy).

By Theorem 1, the ABMC state converges to a unique equilibrium s^* satisfying

the same fixed-point equation $s^* = \mathfrak{T}s^*$. Hence, for each $v_i \in \mathcal{V}$,

$$s_i^* = \min_{\pi:\, v_i \rightsquigarrow v_1} \sum_{(v_p \to v_q) \in \pi} a_{pq} \tag{6.26}$$

i.e., s_i^* equals the minimum cumulative bias along any directed backup path from v_i to the leader v_1 (and $s_1^* = 0$). ■

Remark 5: Upon convergence of each robot's state to the solution of the aforementioned nonlinear equation, the backup path from any robot in $\mathcal{V}_2$ to the leader robot can be constructed by recursively tracing the sequence of backup parent robots.

Remark 6: Theorems 1 and 2 guarantee that all robots' state values converge to their respective backup paths, regardless of their initial states.

6.5 Backup Network Layers

Collectively, the backup paths generated by the ABMC protocol form the backup network layer (Fig. 6.2). The backup network layer allows each robot to receive positional information along multiple paths from the leader, via its primary parents and its backup parents. The backup layer thus provides the structure for a feedback mechanism, enabling independent verification of the accuracy of information flowing along different paths through the swarm. The backup network layer is self-organized using local information from nearby robots and it adaptive to changes in the original graph, thus maintaining the adaptability to current conditions that is crucial for addressing intermittent faults.

6.5.1 Algorithm for backup layer construction

Algorithm 1 details the process of constructing backup paths for an individual robot, for a robot swarm deployed in $\mathbb{R}^2$. The paths constructed throughout a swarm are aggregated to form the backup network layer.

First, for each follower robot $v_i \in \mathcal{V}_2$, the algorithm begins by initializing two sets: one for the minimum-cost backup edge ($\mathcal{E}^{min}$) and one for alternative backup edges ($\mathcal{E}^{alt}$). Note that while the ABMC protocol is originally formulated in contin-

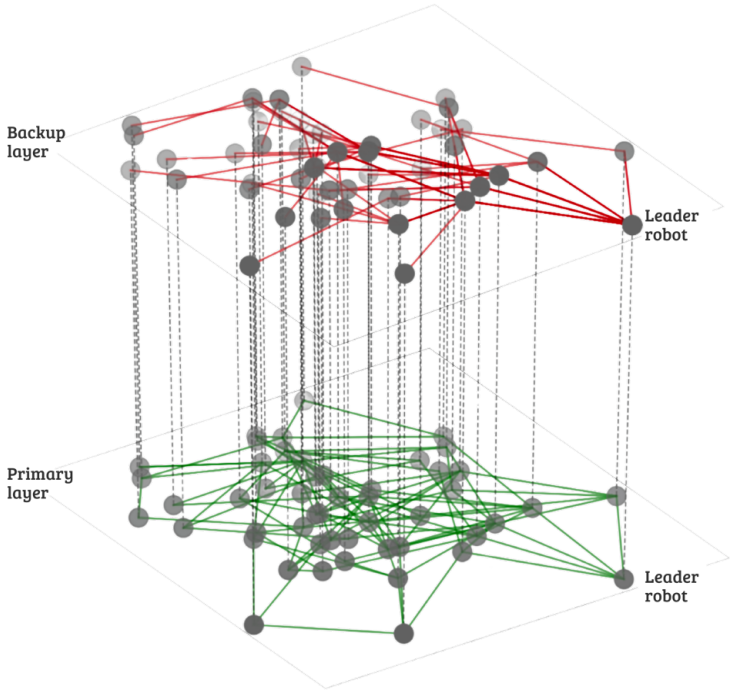


Figure 6.2: Visualization of a backup network layer established by the ABMC protocol for a 50-robot swarm. Each robot (grey circles) independently establishes minimum-cost backup edges (red lines; darker red indicates a higher number of edges between the two respective nodes) that collectively form backup paths to the leader robot. These backup paths collectively form the backup network layer (red) that complements the primary network (green).

uous time (see Eq. (6.3)), in practical implementations the differential equation is discretized. Here, the iteration index k corresponds to discrete time instants $t = k\Delta t$ for a chosen sampling interval Δt, and the update $s_i[k+1]$ approximates the state $s_i(t+\Delta t)$. Define the step size $\int := \Delta t/\eta \in (0, 1]$. We use the forward-Euler update $s_i[k+1] = (1-\int)\, s_i[k] + \int\Big(s_{j^*}[k] + a_{ij^*}[k]\Big)$, where j^* indexes the minimum-cost candidate parent at step k. This convex-combination form preserves nonnegativity for hop-like states when $s_i[0] \geq 0$. Consequently, the convergence criterion $|s_i[k+1] - s_i[k]| < \zeta$, with a predefined threshold $\zeta > 0$, serves as a discrete-time analogue to the continuous-time derivative approaching zero.

Then, the minimum-cost backup path is reconstructed by backtracking from the candidate parent v_{j^*} using the edges stored in $\mathcal{E}^{min}$. Here $v_{j^*} := \underset{v_j \in \mathcal{P}_i^{\text{cand}}[k]}{\arg\min} \Big(s_j[k] + a_{ij}[k]\Big)$ is the minimum-cost candidate parent, and it is thus selected as the backup parent for v_i at iteration k; equivalently, the directed edge (v_i, v_{j^*}) is the link stored for path construction. The algorithm ensures that each robot in the backtracked path is connected to its predecessor via a valid edge from the minimum-cost edge set, $\mathcal{E}^{min}$. In the final iteration, for each robot $v_j \in \mathcal{P}_i^{\text{cand}}[k]$, if $v_j \neq v_{j^*}$ and the difference between the alternative candidate's cost and the minimum-cost path's cost is within an acceptable range, i.e.,

$$\Big|\Big(s_j[k] + a_{ij}[k]\Big) - s_i[k+1]\Big| < \tau \tag{6.27}$$

then the edge (v_i, v_j) is added to the alternative edge set.

The threshold $\tau > 0$ is a predefined parameter that allows slight variations in the path costs, ensuring that an alternative path is considered viable only if its cost is only marginally higher than the minimum-cost path's cost. Subsequently, the alternative paths are obtained by backtracking from each eligible robot v_j using the edges stored in $\mathcal{E}^{alt}$. Upon completion of the backtracking processes, the algorithm stores the resulting minimum-cost backup path $\mathcal{B}_i^{min}$ and the set of alternative backup paths $\mathcal{B}_i^{alt}$ (along with their associated costs and edge sets) for robot v_i in the set $\boldsymbol{\mathcal{B}}_i$.

Remark 7: The discrete next-hop selection $v_{j^\star} \in \underset{v_j \in \mathcal{P}_i^{\text{cand}}[k]}{\arg\min} \Big(s_j[k] + a_{ij}[k]\Big)$ is

Algorithm 1 Construction of backup paths for Robot v_i

Require: η: Convergence rate factor, ρ: Weight adjusting the hierarchy's impact on the bias term, ψ: Weight scaling the contribution of congestion penalty to the overall bias, κ_d: Maximum robot outdegree, $\mathcal{H}_i$: Robot's hierarchy, K: Maximum iterations, τ: Alternative path cost threshold, ζ: Convergence tolerance, r: Communication range, Δt: Sampling interval with $0 < \Delta t/\eta \leq 1$

Ensure: $\boldsymbol{\mathcal{B}}_i$: Set of backup paths (minimum-cost $\mathcal{B}_i^{min}$ and alternatives $\mathcal{B}_i^{alt}$) for v_i.

Initialize:
  $\mathcal{E}^{min} \leftarrow \emptyset$, $\mathcal{E}^{alt} \leftarrow \emptyset$, $\mathcal{B}_i^{min} \leftarrow \emptyset$, $\mathcal{B}_i^{alt} \leftarrow \emptyset$
  $s_i[0] \leftarrow \infty$, converged $\leftarrow$ False, $k \leftarrow 0$
while $k \leq K$ **and** not converged **do**
  Compute the candidate parent set $\mathcal{P}_i^{\text{cand}}[k]$ according to Eq. (6.4)
  for all $v_j \in \mathcal{P}_i^{\text{cand}}[k]$ **do**
    Compute $a_{ij}[k]$ using Eq. (6.5)
  end for
  $v_{j^*} \leftarrow \underset{v_j \in \mathcal{P}_i^{\text{cand}}[k]}{\arg\min} \Big(s_j[k] + a_{ij}[k] \Big)$
  $s_i[k+1] \leftarrow (1 - \int)\, s_i[k] + \int \Big(s_{j^*}[k] + a_{ij^*}[k] \Big)$ {Euler step; preserves $s_i \geq 0$}
  Set backup parent: $\mathcal{P}_i^{\text{back}} \leftarrow v_j^*$
  Let edge $e_{ij^*} \leftarrow (v_i, v_{j^*})$
  $\mathcal{E}^{min} \leftarrow \mathcal{E}^{min} \cup e_{ij^*}$
  converged $\leftarrow \Big(|s_i[k+1] - s_i[k]| < \zeta \Big)$
  $k \leftarrow k + 1$
end while
if *converged* **then**
  $\mathcal{B}_i^{min} \leftarrow$ Backtrack from v_i using the edges in $\mathcal{E}^{min}$ (cf. Remark 5)
end if
for all $v_j \in \mathcal{P}_i^{\text{cand}}[k]$ such that $v_j \neq v_{j^*}$ and $\Big| (s_j[k] + a_{ij}[k]) - s_i[k+1] \Big| < \tau$ **do**
  Let edge $e_{ij} \leftarrow (v_i, v_j)$
  $\mathcal{E}^{alt} \leftarrow \mathcal{E}^{alt} \cup e_{ij}$
  $\mathcal{B}_i^{alt} \leftarrow$ Backtrack from v_i using the edges in $\mathcal{E}^{alt}$
end for
$\boldsymbol{\mathcal{B}}_i \leftarrow$ {'minimum-cost': $\mathcal{B}_i^{min}$, 'alternatives': $\mathcal{B}_i^{alt}$}
return $\boldsymbol{\mathcal{B}}_i$

set-valued under ties or near-ties, which can lead to chattering even when $s_i[k]$ has settled. This can be suppressed using a hysteresis policy (often used to suppress chattering and limit rapid switching (Hespanha et al., 2003; Lin and Antsaklis, 2009)): in short, retain the current parent unless a candidate provides a clearly better local cost, or the improvement is consistently observed over several updates.

6.5.2 Properties of backup layer construction

Runtime: The time complexity of Algorithm 1 for a single robot v_i is primarily determined by the main loop, which iterates up to K times, where K is a constant representing the maximum number of iterations. Each iteration involves several key steps: candidate parent selection, computation of bias terms and state updates, identification of the minimum-cost parent, and a convergence check.

Robot selection is efficiently handled using a grid-based method (Thrun, 2002), which runs in $O(1 + |\mathcal{P}_i^{\text{cand}}|)$ time. For each node $v_j \in \mathcal{P}_i^{\text{cand}}$, the computation of the bias $a_{ij}[k]$ and the update of the state value $s_i[k+1]$ are constant-time operations, yielding a total per-iteration cost of $O(|\mathcal{P}_i^{\text{cand}}|)$ for these steps. Identifying the minimum-cost parent v_j^* (i.e., the one that minimizes the cost $s_j[k] + a_{ij}[k]$) requires scanning through all candidate parents, which also takes $O(|\mathcal{P}_i^{\text{cand}}|)$. The convergence check—determining whether $|s_i[k+1] - s_i[k]|$ falls below a predefined threshold ζ—is an $O(1)$ operation. Hence, each iteration runs in $O(1 + |\mathcal{P}_i^{\text{cand}}|)$ time. Since K is constant, the cumulative time complexity over the main loop is $O(K(1 + |\mathcal{P}_i^{\text{cand}}|)) = O(|\mathcal{P}_i^{\text{cand}}|)$.

Once convergence is achieved, the algorithm backtracks the minimum-cost path from the selected parent v_j^* using the stored edges $\mathcal{E}^{min}$. This backtracking process has a time complexity of $O(L)$, where L is the length of the path (i.e., the number of hops) to the leader robot. Additionally, the algorithm backtracks alternative paths from each eligible robot $v_j \in \mathcal{P}_i^{\text{cand}}$. For each such robot—if it satisfies the condition in Eq. (6.27), an alternative path is backtracked using the stored edges $\mathcal{E}^{alt}$. As there may be up to $|\mathcal{P}_i^{\text{cand}}|$ alternative paths, each requiring $O(L)$ time, this step has a complexity of $O(|\mathcal{P}_i^{\text{cand}}|L)$.

Combining all steps, the overall time complexity—including the main loop and the backtracking processes—is $O(1+|\mathcal{P}_i^{\text{cand}}|+L+|\mathcal{P}_i^{\text{cand}}|L)$. Since constant terms and lower-order terms are absorbed by the dominant factor, and given that L is generally small compared to the total number of robots, the final complexity simplifies to $O(|\mathcal{P}_i^{\text{cand}}|L)$.

Memory space requirements: In terms of the memory space required for a single robot to run Algorithm 1, storing the minimum-cost path $\mathcal{B}_i^{min}$ requires $O(L)$ space, while storing all alternative paths $\mathcal{B}_i^{alt}$ requires up to $O(|\mathcal{P}_i^{\text{cand}}|L)$ space, since there may be as many as $|\mathcal{P}_i^{\text{cand}}|$ alternative paths (each of length L). Thus, the total space complexity for storing all paths is $O(|\mathcal{P}_i^{\text{cand}}|L)$.

Efficiency: The efficiency of Algorithm 1 for a single robot can be influenced by the spatial layout of the robots in the formation. In sparser areas, where the number of robots in in the communication range r of robot v_i is much smaller than the total number of robots in the swarm, the algorithm performs more efficiently in both time and space, making it suitable for real-time applications. In denser formations, where the number of robots in $|\mathcal{P}_i^{\text{cand}}|$ increases w.r.t. the total number of robots in the swarm, performance will be more affected by the number of candidate parents

6.6 Proactive–Reactive Detection and Mitigation of Intermittent Faults

Intermittent faults cause transient deviations (biases, offset errors, noise), and thus can perturb the statistical properties of transmitted data, particularly the mean and variance (Muhammed and Shaikh, 2017). At each time t, robot v_i in $\mathbb{R}^2$ receives relative position data from: (1) a path of direct parent robot $v_j \in \mathcal{P}_i$ that may exhibit offset faults, and (2) a fault-free backup path $\mathcal{B}_i[b] \in \boldsymbol{\mathcal{B}}_i$ from a leader robot.

We model offset faults as:

$$f_{x_{ij}}(t) = o_{x_{ij}}\, s(t)\, w(t), \quad f_{y_{ij}}(t) = o_{y_{ij}}\, s(t)\, w(t) \tag{6.28}$$

where $o_{x_{ij}}$, $o_{y_{ij}}$ are constant offsets, s is a Bernoulli random variable with probability P_f, and w is a function indicating fault duration.

The potentially faulty data $\tilde{\mathbf{q}}_{ij}(t) = [\, x_{ij}(t) + f_{x_{ij}}(t),\, y_{ij}(t) + f_{y_{ij}}(t)\,]^{\mathsf{T}}$ follows a Gaussian distribution:

$$\tilde{\mathbf{q}}_{ij}(t) \sim \mathfrak{N}\left(\begin{bmatrix} x_{ij}(t) + P_f \times o_{x_{ij}} \\ y_{ij}(t) + P_f \times o_{y_{ij}} \end{bmatrix}, \begin{bmatrix} P_f \times (1 - P_f) \times o_{x_{ij}}^2 & 0 \\ 0 & P_f \times (1 - P_f) \times o_{y_{ij}}^2 \end{bmatrix} \right) \tag{6.29}$$

where $\mathfrak{N}$ denotes the Gaussian (normal) distribution.

Conversely, the fault-free backup data is:

$$\mathbf{q}_{ij}(t) = \begin{bmatrix} x_{ij}(t) \\ y_{ij}(t) \end{bmatrix} \sim \mathfrak{N}\Big(\begin{bmatrix} x_{ij}(t) \\ y_{ij}(t) \end{bmatrix}, \mathbf{0} \Big) \tag{6.30}$$

Each relative-position packet transmitted from robot v_j to v_i traverses a binary-symmetric channel with bit-error probability P_e. Equivalently, we approximate the effect of bit errors as an additional additive noise term,

$$\mathbf{q}_{ij}^{\mathrm{rx}}(t) = \mathbf{q}_{ij}(t) + \boldsymbol{\eta}_{ij}(t), \quad \boldsymbol{\eta}_{ij}(t) \sim \mathfrak{N}\big(\mathbf{0},\, \sigma_e^2(P_e)\mathbf{I}\big) \tag{6.31}$$

where the noise variance σ_e^2 is chosen proportional to P_e to reflect the expected distortion caused by random bit flips.

In our scenario, in which backup paths to the leader have already been constructed, the detection task can be reduced to checking whether the distribution of primary data differs significantly from that of the backup data. To check this, we adopt a Likelihood Ratio (LR) test for the data from the primary parents compared to the data from the backup parents, contrasting two hypotheses:

- **Null hypothesis** H_0: the data are consistent with the backup (no fault present).
- **Alternative hypothesis** H_1: the data distribution is perturbed (fault present).

Note that we use the *log*-Likelihood Ratio (LLR) for numerical stability and additivity, which yields the same decisions as the LR because $\log(\cdot)$ is monotone. Each

robot computes LLR values for its parents and uses them to decide whether a fault is present and, if so, whether to switch to a backup path.

6.6.1 Algorithm for fault detection and mitigation

First, Algorithm 2 collects N data points from both the primary parents $\mathcal{P}_i$ and the backup parents $\mathcal{P}_i^{\text{back}}$ of robot v_i, and the sample means and covariances are calculated for these data points. Then, *log*-Likelihood Ratio (LLR) tests are computed for robot v_i across its all backup paths $\mathcal{B}_i[b] \in \boldsymbol{\mathcal{B}}_i$, comparing the likelihood of the potentially faulty data under the Alternative hypothesis (i.e., faulty hypothesis) against the Null hypothesis (i.e., fault-free hypothesis). To calculate the LLRs, we first calculate LR. Let $\tilde{\mathbf{Q}}_{ij} \triangleq \{\tilde{\mathbf{q}}_{ij}[k] : k = 1, 2, \ldots, N\}$ denote the N samples from the potentially faulty parent robot, and $\mathbf{Q}_{ij}[b] \triangleq \{\mathbf{q}_{ij}[k] : k = 1, 2, \ldots, N\}$ the N samples from the fault-free backup path $\mathcal{B}_i[b] \in \boldsymbol{\mathcal{B}}_i$.

We first compute sample means. For the parent robot path:

$$\boldsymbol{\mu}_{\tilde{\mathbf{Q}}_{ij}} = \frac{1}{N}\sum_{k=1}^{N} \tilde{\mathbf{q}}_{ij}[k] = \left[\mu_{\tilde{\mathbf{X}}_{ij}},\, \mu_{\tilde{\mathbf{Y}}_{ij}}\right] \tag{6.32}$$

where

$$\mu_{\tilde{\mathbf{X}}_{ij}} = \frac{1}{N}\sum_{k=1}^{N} \tilde{x}_{ij}[k], \quad \mu_{\tilde{\mathbf{Y}}_{ij}} = \frac{1}{N}\sum_{k=1}^{N} \tilde{y}_{ij}[k]$$

Similarly, for the fault-free backup path:

$$\boldsymbol{\mu}_{\mathbf{Q}_{ij}[b]} = \frac{1}{N}\sum_{k=1}^{N} \mathbf{q}_{ij}[k] = \left[\mu_{\mathbf{X}_{ij}[b]},\, \mu_{\mathbf{Y}_{ij}[b]}\right] \tag{6.33}$$

where

$$\mu_{\mathbf{X}_{ij}[b]} = \frac{1}{N}\sum_{k=1}^{N} x_{ij}[k], \quad \mu_{\mathbf{Y}_{ij}[b]} = \frac{1}{N}\sum_{k=1}^{N} y_{ij}[k]$$

Next, the sample covariance for the parent robot path is

$$\boldsymbol{\Sigma}_{\tilde{\mathbf{Q}}_{ij}} = \frac{1}{N-1}\sum_{k=1}^{N}\left(\tilde{\mathbf{q}}_{ij}[k] - \boldsymbol{\mu}_{\tilde{\mathbf{Q}}_{ij}}\right)\left(\tilde{\mathbf{q}}_{ij}[k] - \boldsymbol{\mu}_{\tilde{\mathbf{Q}}_{ij}}\right)^T \tag{6.34}$$

Algorithm 2 *Proactive–reactive* fault detection and mitigation

Require: $\boldsymbol{\mathcal{B}}_i$: Set of backup paths for robot v_i to the leader robot (from Algorithm 1), including: $\mathcal{B}_i^{min}$: Minimum-cost backup path (a list of (robot, cost) tuples), and $\mathcal{B}_i^{alt}$: Alternative backup paths (each a list of (robot, cost) tuples); $\tilde{\mathbf{Q}}_{ij} \triangleq \{\tilde{\mathbf{q}}_{ij}[k] : k = 1, 2, \ldots, N\}$: Potentially faulty data set includes N number $\tilde{\mathbf{q}}_{ij}$ data; $\mathbf{Q}_{ij}[b] \triangleq \{\mathbf{q}_{ij}[k] : k = 1, 2, \ldots, N\}$: Fault-free data set includes N number $\mathbf{q}_{ij}$ data from backup path $\mathcal{B}_i[b] \in \boldsymbol{\mathcal{B}}_i$; *use_backup*: Boolean flag indicating if the backup path is in use; T^{lock}: Timer to prevent rapid switching; T^{dur}: A positive real number defining the minimum time between decision changes; Δt: Sampling interval

Ensure: Fault detection decision for robot v_i

for all parent robots $v_j \in \mathcal{P}_i$ **do**
  $\mathbf{LLR}_{ij} \leftarrow \emptyset$
  for all backup path $\mathcal{B}_i[b] \in \boldsymbol{\mathcal{B}}_i$ **do**
    $\boldsymbol{\mu}_{\tilde{\mathbf{Q}}_{ij}} \leftarrow$Eq.(6.32), $\boldsymbol{\mu}_{\mathbf{Q}_{ij}[b]} \leftarrow$ Eq.(6.33),
    $\boldsymbol{\Sigma}_{\tilde{\mathbf{Q}}_{ij}} \leftarrow$ Eq.(6.34), and $\boldsymbol{\Sigma}_{\mathbf{Q}_{ij}[b]} \leftarrow$ Eq.(6.35)
    $LLR_{ij}[b] \leftarrow$ Eq. (6.38)
    $\mathbf{LLR}_{ij} \leftarrow \mathbf{LLR}_{ij} \cup \{LLR_{ij}[b]\}$
  end for
  $\mathcal{Q}_1 \leftarrow$ Eq.(6.40) and $\mathcal{Q}_3 \leftarrow$ Eq.(6.41)
  $\lambda_{ij}^{IQR} \leftarrow$ Eq.(6.42) and $\lambda_{ij} \leftarrow$ Eq.(6.39)
  $\lambda_{ij}^{\text{recover}} \leftarrow$ Eq.(6.43)
  $n^{faulty} \leftarrow \sum_{\mathcal{B}_i[b] \in \mathcal{B}_i} \mathcal{I}[b]$
  if $n^{faulty} \geq \Gamma$ **and** $T^{\text{lock}} \leq 0$ **then**
    $\boldsymbol{\mathcal{B}}_i^{cand} \leftarrow \{\mathcal{B}_i[b] \in \boldsymbol{\mathcal{B}}_i \mid LLR_{ij}[b] > \lambda_{ij}\}$
    if $\mathcal{B}_i^{min} \in \boldsymbol{\mathcal{B}}_i^{cand}$ **then**
      $\mathcal{B}_i[b^*] \leftarrow \mathcal{B}_i^{min}$
    else
      $\mathcal{B}_i[b^*] \leftarrow \arg\max_{\mathcal{B}_i[b] \in \boldsymbol{\mathcal{B}}_i^{cand}} LLR_{ij}[b]$
    end if
    use_backup $\leftarrow$ **True**
    $T^{\text{lock}} \leftarrow T^{\text{dur}}$
  else if *use_backup* **and** $\lambda_{ij}^{med} < \lambda_{ij}^{\text{recover}}$ **and** $T^{\text{lock}} \leq 0$ **then**
    use_backup $\leftarrow$ **False**
    $T^{\text{lock}} \leftarrow T^{\text{dur}}$
  end if
  $T^{\text{lock}} \leftarrow \max(0, T^{\text{lock}} - \Delta t)$
end for
return Fault detected or no fault – Continue with chosen data source

For the fault-free backup path, we assume zero covariance:

$$\boldsymbol{\Sigma}_{\mathbf{Q}_{ij}[b]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.35}$$

The LR compares the probability of the observed parent robot data $\tilde{\mathbf{Q}}_{ij}$ under the fault hypothesis H_1 versus the no-fault hypothesis H_0. Under H_1, the Gaussian PDF parameters (mean and covariance) are estimated directly from the parent data $\tilde{\mathbf{Q}}_{ij}$. Under H_0, the Gaussian PDF is defined by the parameters estimated from the fault-free backup path data $\mathbf{Q}_{ij}[b]$. In this formulation, the backup data do not appear explicitly in the likelihood product; rather, they determine the nominal model parameters.

$$LR_{ij}[b] = \frac{\Pr\{\tilde{\mathbf{Q}}_{ij} \mid H_1\}}{\Pr\{\tilde{\mathbf{Q}}_{ij} \mid H_0\}} \tag{6.36}$$

where

$$\Pr\{\tilde{\mathbf{Q}}_{ij} \mid H_1\} = \prod_{k=1}^{N} f\Big(\tilde{\mathbf{q}}_{ij}[k];\, \boldsymbol{\mu}_{\tilde{\mathbf{Q}}_{ij}},\, \boldsymbol{\Sigma}_{\tilde{\mathbf{Q}}_{ij}}\Big)$$

and

$$\Pr\{\tilde{\mathbf{Q}}_{ij} \mid H_0\} = \prod_{k=1}^{N} f\Big(\tilde{\mathbf{q}}_{ij}[k];\, \boldsymbol{\mu}_{\mathbf{Q}_{ij}[b]},\, \boldsymbol{\Sigma}_{\mathbf{Q}_{ij}[b]}\Big).$$

Here, $f(\cdot; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ denotes the Gaussian PDF with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$.

Substituting these PDFs, the closed-form LR is

$$\begin{aligned} LR_{ij}[b] = \left(\frac{|\boldsymbol{\Sigma}_{\mathbf{Q}_{ij}[b]}|}{|\boldsymbol{\Sigma}_{\tilde{\mathbf{Q}}_{ij}}|}\right)^{N/2} \exp\Bigg(-\frac{1}{2}\sum_{k=1}^{N}\Big[\\ (\tilde{\mathbf{q}}_{ij}[k] - \boldsymbol{\mu}_{\tilde{\mathbf{Q}}_{ij}})^T \boldsymbol{\Sigma}_{\tilde{\mathbf{Q}}_{ij}}^{-1} (\tilde{\mathbf{q}}_{ij}[k] - \boldsymbol{\mu}_{\tilde{\mathbf{Q}}_{ij}}) \\ - (\mathbf{q}_{ij}[k] - \boldsymbol{\mu}_{\mathbf{Q}_{ij}[b]})^T \boldsymbol{\Sigma}_{\mathbf{Q}_{ij}[b]}^{-1} (\mathbf{q}_{ij}[k] - \boldsymbol{\mu}_{\mathbf{Q}_{ij}[b]}) \Big] \Bigg) \end{aligned} \tag{6.37}$$

The use of a zero covariance matrix for the fault-free backup path may lead to numerical issues. In particular, a zero covariance results in a zero determinant and a non-invertible matrix, which renders the Gaussian likelihood calculations undefined.

To avoid these singularities and account for minimal inherent noise, it is common practice to introduce a small positive constant to the diagonal entries of the covariance matrix.

The log-likelihood ratio (LLR) is:

$$\begin{aligned} LLR_{ij}[b] = \frac{N}{2} \ln \left(\frac{|\mathbf{\Sigma}_{\mathbf{Q}_{ij}[b]}|}{|\mathbf{\Sigma}_{\tilde{\mathbf{Q}}_{ij}}|} \right) - \frac{1}{2} \sum_{k=1}^{N} \Big[\\ (\tilde{\mathbf{q}}_{ij}[k] - \boldsymbol{\mu}_{\tilde{\mathbf{Q}}_{ij}})^T \mathbf{\Sigma}_{\tilde{\mathbf{Q}}_{ij}}^{-1} (\tilde{\mathbf{q}}_{ij}[k] - \boldsymbol{\mu}_{\tilde{\mathbf{Q}}_{ij}}) \\ - (\mathbf{q}_{ij}[k] - \boldsymbol{\mu}_{\mathbf{Q}_{ij}[b]})^T \mathbf{\Sigma}_{\mathbf{Q}_{ij}[b]}^{-1} (\mathbf{q}_{ij}[k] - \boldsymbol{\mu}_{\mathbf{Q}_{ij}[b]}) \Big] \end{aligned} \tag{6.38}$$

Because ln of small determinant values can cause further numerical issues, one can impose a lower bound on $|\mathbf{\Sigma}_{\mathbf{Q}_{ij}[b]}|$ to avoid excessive underflow in practical implementations.

After calculating the LLRs, they are stored in a set, $\mathbf{LLR}_{ij} = \{LLR_{ij}[b] \mid \mathcal{B}_i[b] \in \boldsymbol{\mathcal{B}}_i\}$. Next, the algorithm calculates two thresholds.

Detection threshold: λ_{ij} is computed based on the median and interquartile range (IQR) of N LLR values sorted in ascending order, denoted as $\mathbf{LLR}^{sorted}$. This threshold ensures robust error detection and is defined as

$$\lambda_{ij} = \lambda_{ij}^{med} + 1.5 \times \lambda_{ij}^{IQR}, \tag{6.39}$$

where λ^{med} is the median value and λ^{IQR} is the interquartile range computed as the difference between the 75th percentile ($\mathcal{Q}_3$) and the 25th percentile ($\mathcal{Q}_1$) of the LLR values. Here, let m = $|\mathbf{LLR}_{ij}|$ denote the total number of LLR values. Then, the 25th and 75th percentiles are given by

$$\mathcal{Q}_1 = \mathbf{LLR}_{ij}^{sorted}\Big(\lceil 0.25 \times \mathrm{m} \rceil \Big), \tag{6.40}$$

$$\mathcal{Q}_3 = \mathbf{LLR}_{ij}^{sorted}\Big(\lceil 0.75 \times \mathrm{m} \rceil \Big). \tag{6.41}$$

Finally, the interquartile range is

$$\lambda_{ij}^{IQR} = \mathcal{Q}_3 - \mathcal{Q}_1. \tag{6.42}$$

Recovery threshold: $\lambda_{ij}^{\text{recover}}$ is computed using the minimum and maximum LLR values within the recent N LLR values, with a tunable factor θ. This ensures that the system only switches back to the primary path when the LLR values have stabilized across a narrow range. The recovery threshold is defined as

$$\lambda_{ij}^{\text{recover}} = \lambda_{ij}^{\min} + \theta \times (\lambda_{ij}^{\max} - \lambda_{ij}^{\min}) \tag{6.43}$$

where $\lambda^{\min}$ is the minimum LLR value and $\lambda^{\max}$ is the maximum LLR value within the recent N LLR values. This formulation takes into account the spread of the LLR values, ensuring that switching back to the primary path occurs only when the system has sufficiently stabilized.

After calculating the thresholds, the presence of a fault is confirmed based on the number of backup paths whose LLR exceeds the detection threshold λ_{ij}. A binary indicator variable $\mathcal{I}$ is set for each backup path based on whether the corresponding LLR exceeds λ_{ij}:

$$\mathcal{I}[b] = \begin{cases} 1, & \text{if } LLR_{ij}[b] > \lambda_{ij} \\ 0, & \text{otherwise} \end{cases} \tag{6.44}$$

A fault is declared if the number of paths indicating a fault, n^{faulty}, meets or exceeds a majority threshold Γ, typically set to the ceiling of half the number of backup paths:

$$n^{faulty} \geq \Gamma$$

Upon detecting a fault, the algorithm identifies a set of candidate backup paths, $\boldsymbol{\mathcal{B}}_i^{cand}$, that have LLRs exceeding the detection threshold λ_{ij}. The minimum-cost backup path $\mathcal{B}_i^{min}$ is selected if it is among the candidates; otherwise, an alternative backup path $\mathcal{B}_i^{alt}$ with the highest LLR is chosen.

Switching back to the primary path occurs when the fault condition resolves.

Detection of fault resolution is based on comparing the median LLR value, $\lambda_{ij}^{\text{med}}$, with the recovery threshold $\lambda_{ij}^{\text{recover}}$. If the median LLR value drops below the recovery threshold, it indicates that the system has stabilized, and it is safe to revert to the primary path. This mechanism ensures that the switch back to the primary path happens only when the fault has truly subsided, preventing premature switching in the presence of temporary fluctuations in the LLR values.

To ensure stability, a lock-in timer T^{lock} prevents frequent switching between the primary and backup paths. After switching to a path, the system enforces a waiting period (lock-in time) before reevaluating the conditions for switching back, stabilizing the decision-making process.

6.6.2 Properties of fault detection and mitigation

Runtime: The time complexity of Algorithm 2 is primarily influenced by the computation of sample means, covariance matrices, the LLR values, and sorting the LLRs. For each parent robot, these calculations are performed for every backup path and involve operations over the N data points in each path. Computing the sample mean $\boldsymbol{\mu}$ of a dataset with N points in d-dimensional space ($d = 2$ here) requires $O(Nd)$ time; since d is constant, this is $O(N)$ per dataset. Similarly, computing the sample covariance $\boldsymbol{\Sigma}$ takes $O(Nd^2) = O(N)$ (again d is constant). Thus, for each backup path, the total to compute mean+covariance is $O(N)$.

The per-path LLR can be evaluated in $O(1)$ *if* we use the already computed sufficient statistics (mean, covariance, and N) rather than re-summing over samples (cf. Eq. (6.38) written in terms of sufficient statistics). Sorting the $|\boldsymbol{\mathcal{B}}_i|$ LLR values costs $O(|\boldsymbol{\mathcal{B}}_i| \log |\boldsymbol{\mathcal{B}}_i|)$. Therefore, for a single parent v_j, the total time is $O\Big(N|\boldsymbol{\mathcal{B}}_i|\Big) + O\Big(|\boldsymbol{\mathcal{B}}_i| \log |\boldsymbol{\mathcal{B}}_i|\Big)$.

If the statistics of the potentially faulty parent stream $\tilde{\mathbf{Q}}_{ij}$ are reused across all backup paths (computed once), the bound remains the same. For completeness, multiplying by the in-degree δ_i^{in} gives a per-robot bound $O\Big(\delta_i^{\text{in}}(N|\boldsymbol{\mathcal{B}}_i| + |\boldsymbol{\mathcal{B}}_i| \log |\boldsymbol{\mathcal{B}}_i|)\Big)$; in our HHC setting $\delta_i^{\text{in}} \in \{1, 2\}$.

Memory space requirements: The space complexity is dominated by the

storage requirements for the data points, covariance matrices, and LLR values. Each backup path requires storage for N data points of dimension d, totaling $O(Nd)$ space per path. For all backup paths, this amounts to $O(Nd|\boldsymbol{\mathcal{B}}_i|)$. With d being constant, this simplifies to $O(N|\boldsymbol{\mathcal{B}}_i|)$. Each covariance matrix and mean vector requires $O(d^2)$ and $O(d)$ space, respectively, per dataset. Since d is constant, the total space for all backup paths is $O(|\boldsymbol{\mathcal{B}}_i|)$, which is negligible compared to the data storage. Storing the LLR values for all backup paths requires $O(|\boldsymbol{\mathcal{B}}_i|)$ space. Therefore, the overall space complexity is: $O(N|\boldsymbol{\mathcal{B}}_i|)$

Efficiency: Given a limited number of backup paths $|\boldsymbol{\mathcal{B}}_i|$ as well as reasonable values of N, the algorithm remains efficient and suitable for real-time fault detection. The linear scaling with N in both time and space complexities indicates that the algorithm can handle larger datasets without significant performance penalties.

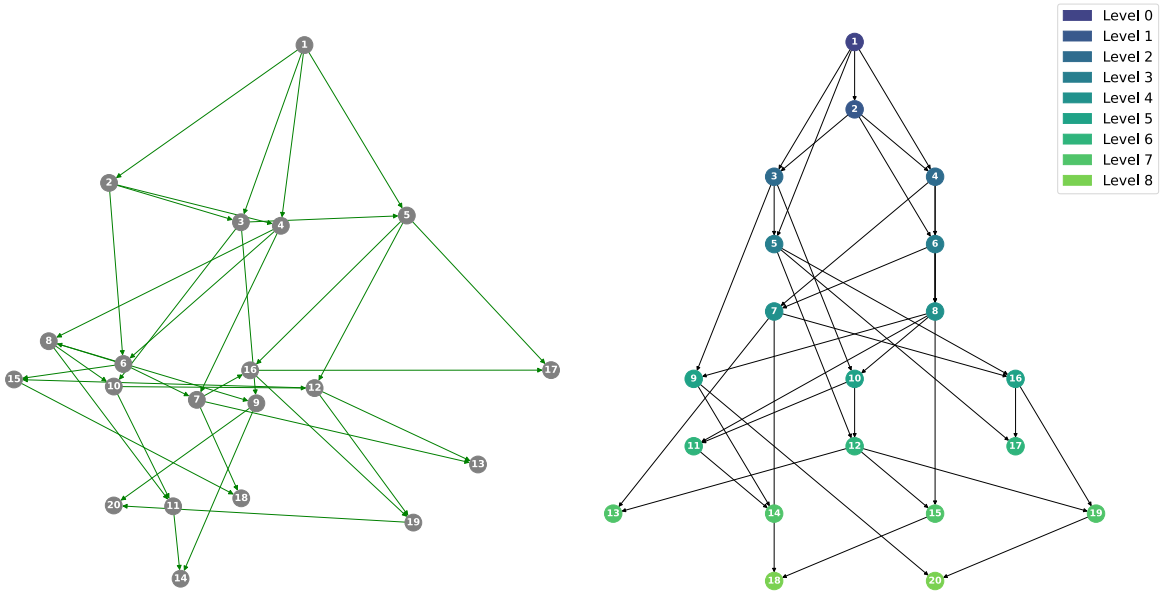
6.7 Validation

Using experimental results in simulation, we first demonstrate the adaptability of the backup network layer to reconfigurations in the original graph. This is shown in a scenario with a static leader in which a high proportion of the swarm fails permanently, resulting in a substantial reconfiguration of the original HHC-constructed graph. Second, we demonstrate our *proactive-reactive* fault tolerance strategy for detecting and mitigating intermittent faults in an HHC-constructed formation with a moving leader, and compare its detection of simple and complex IFs to that of a centralized benchmark. Finally, we demonstrate IF detection and mitigation using our *proactive-reactive* method in a proof-of-concept swarm of 200 robots.

6.7.1 Example scenario 1: Permanent faults

We consider a group of 20 robots in a 2-dimensional space, in an example HHC-constructed graph with eight hierarchy levels. In Fig. 6.3(a), the HHC-constructed graph and the spatial layout of the formation are shown, with the robots represented as nodes. The HHC-constructed graph was generated randomly, but with the leader only allowed to have up to four children and each follower up to three. The

hierarchical relationships among the robots are illustrated in Fig. 6.3(b).



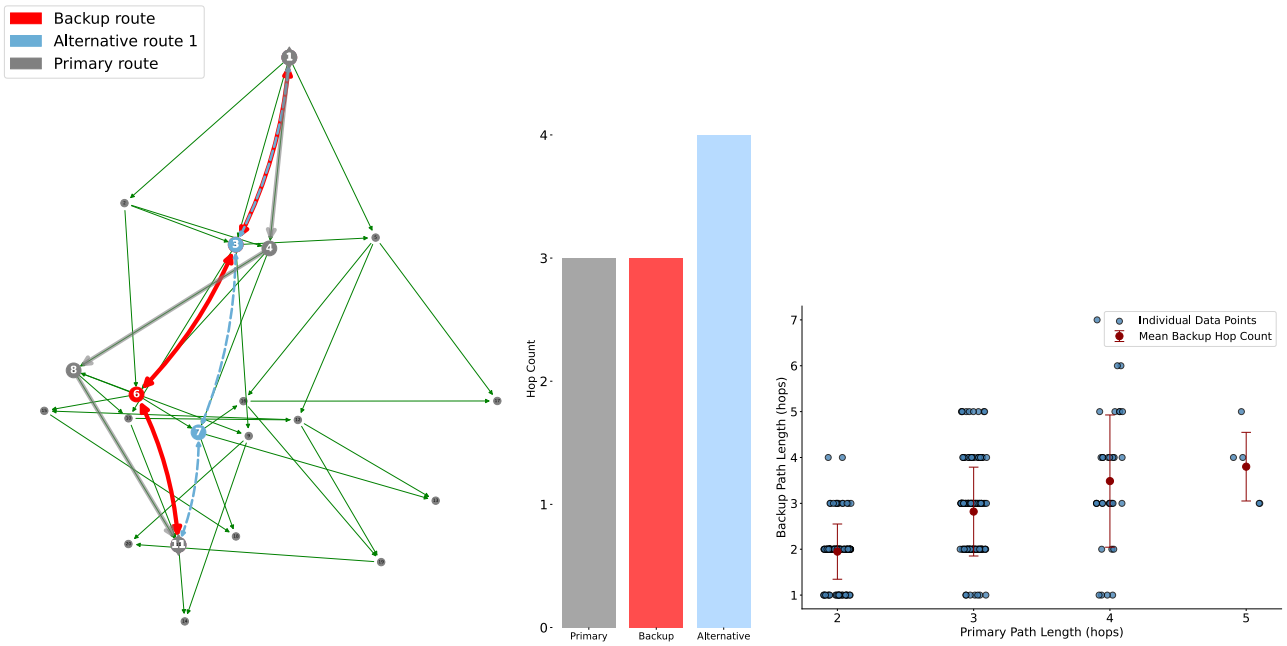
(a) (b)

Figure 6.3: **Example scenario 1: Example primary network.** (a) A randomly formed HHC-constructed graph of 20 robots. Node 1 is the leader robot. (b) The hierarchical structure underlying the formation in (a), where robots are arranged in levels based on their hop count from the leader (Level 0). Each robot is assigned a level that is one higher than the highest level among its parent robots, and information flows from higher levels (starting from Level 0) to lower levels.

In Fig. 6.4(a), the simulation results for node 11 (with parameters configured as $\eta = 0.1$, $\kappa_d = 6$, and $r = 2$ meters) are shown as a representative example, alongside its corresponding path in the primary network. The simulation results established one minimum-cost and one alternative backup path for node 11. In Fig. 6.4(b), it is shown that both the primary path and the minimum-cost backup path have a hop count of 3, while the alternative backup path has a hop count of 4. Thus, in the event of a fault in the primary path, the minimum-cost backup path can be used without increasing the number of hops, minimizing any potential latency increase or disruption to network performance. Although the alternative

backup path involves one additional hop, it can still serve as a fail-safe to maintain communication. In Fig. 6.4(c) we group the backup-path hop counts according to their respective primary-path hop counts. For primary paths of 2 and 3 hops, the mean backup-path hop count was $\approx 2.0 \pm 0.2$ and $\approx 3.1 \pm 0.3$, respectively, showing that approximately 90% of robots found an equally short or only one-hop-longer backup path. For primary paths of 4 and 5 hops, the mean was $\approx 4.3 \pm 0.5$ and $\approx 5.0 \pm 0.7$ hops, respectively, showing larger variability in longer paths.

The adaptability of the ABMC protocol to network changes due to random, permanent robot failures is shown in Fig. 6.5. Following the random removal of 9 out of 20 robots (45% of the swarm) in a representative example trial, the primary network reconfigures using HHC-reconstruction. The new primary network is shown in Fig. 6.5(a). Then, to construct new backup paths following the reconfiguration, each robot re-executes the ABMC protocol to update its backup edges. In Fig. 6.5(b), the simulation results for the new HHC-reconfigured graph are presented, comparing the minimum-cost and alternative backup paths for node 11 to its path in the primary network. In this case, the primary path had a hop count of 2, while both the minimum-cost and alternative backup paths had a hop count of 3. In Fig. 6.5(c), we plot the utilized backup-path hop counts according to their respective primary-path hop counts, grouped by failure rate (5–9 robots, or 25–45% of the swarm, are randomly removed in each trial). For primary paths of 2 hops, the mean backup-path hop count increases only slightly from ≈ 2.1 hops at 5 failures to ≈ 2.2 hops at 9 failures, with σ only increasing from ≈ 0.2 to ≈ 0.35. The failure rate has a moderately increasing effect on primary paths of 3 and 4 hops, with an occasional 1 or 2 hop increase occurring for 3-hop primary paths and an occasional 2 or 3 hop increase for 4-hop primary paths. For primary paths of 5 hops, the failure rate has a much more marked effect on the backup-path hop count, but the relatively small number of samples at this length might contribute to the spread. Overall, even with up to 45% of robots removed, the ABMC protocol always finds a backup path, and the backup paths are usually within 1–2 hops of the primary path (increases of 3 hops occur only occasionally). Mean hop counts increase only modestly with the failure rate, until reaching primary paths of 5 hops (for which there are few samples).



(a) (b) (c)

Figure 6.4: **Example scenario 1: Proactive construction of backup layer before permanent faults occur.** (a) ABMC protocol performance for the formation shown in Fig. 6.3. The shortest path to robot 11 (chosen as a representative example robot) in the primary network is highlighted in gray. The minimum-cost backup path to the leader robot generated by the ABMC protocol for robot 11 is highlighted in red, while the alternative backup path is highlighted in blue. The shortest primary path follows the sequence of robots $1 \to 4 \to 8 \to 11$. In this notation, $\to$ the single arrow indicates a directed link, specifying the intended direction of communication along the primary path. The minimum-cost backup path is $1 \to 3 \to 6 \to 11$, and an alternative path is $1 \to 3 \to 7 \to 11$. (b) Comparison of the shortest path in the primary network for robot 11 and the number of hops for the minimum-cost and alternative backup paths. (c) Backup path performance aggregated by primary path length across 20 independent ABMC trials on distinct graph realizations. For each primary-path hop count (2–5 hops), the mean backup-path hop count (of all follower robots, all 20 trials) is shown in red, with error bars denoting $\pm 1\sigma$. The individual backup-path hop counts are shown in blue. Note that path lengths of 1 hop are omitted, since only follower faults are considered in this study, and thus direct connections to the leader do not require a backup path.

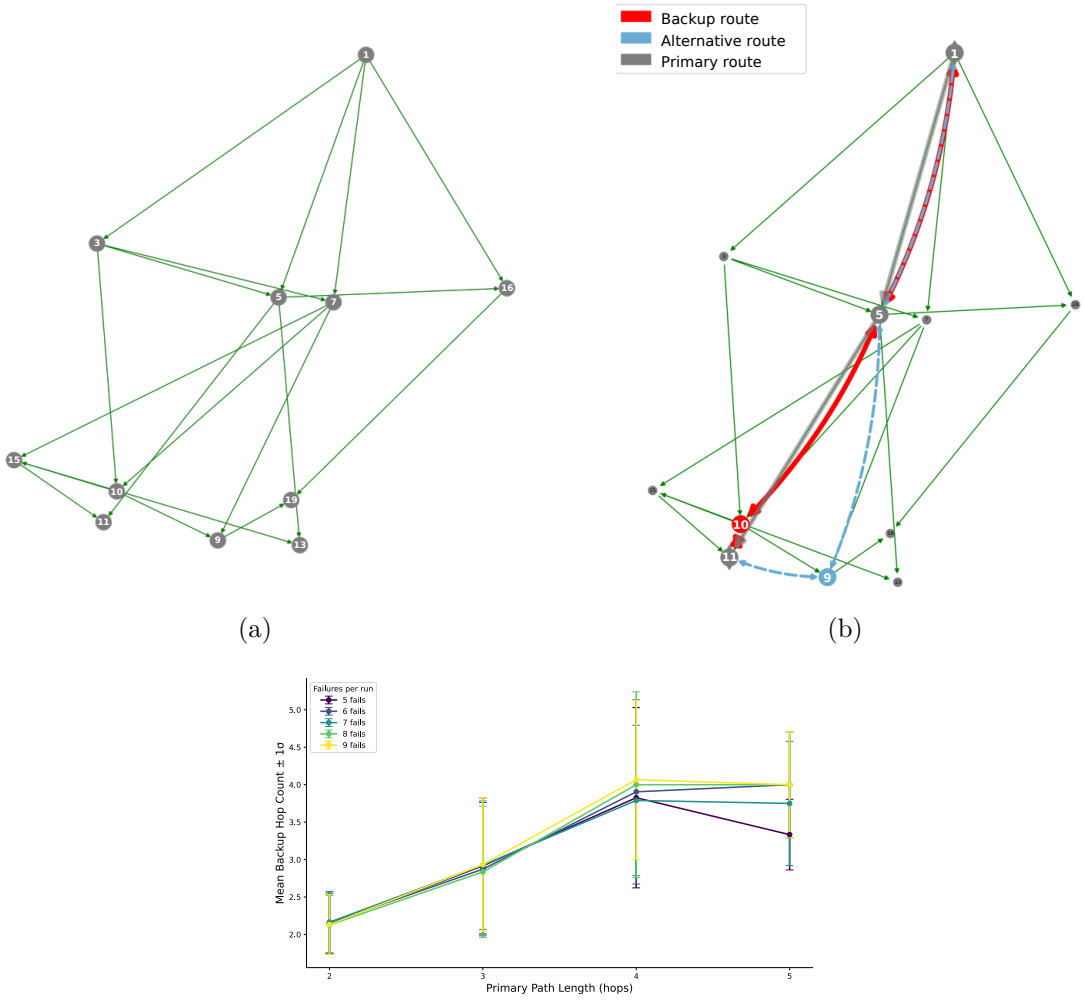


Figure 6.5: **Example scenario 1: Backup paths used when permanent faults occur.** (a) Reconfigured robot formation following the failures in the formation shown in Fig. 6.3. Robots 12, 17, 8, 2, 4, 18, 14, 6, and 20 have been removed in this representative example case. (b) In the reconfigured formation, the new shortest path for robot 11 is shown in gray, while the minimum-cost backup and an alternative path are depicted in red and blue, respectively. The revised primary path now follows the sequence $1 \rightarrow 5 \rightarrow 11$, with the backup path as $1 \rightarrow 5 \rightarrow 10 \rightarrow 11$ and the alternative path as $1 \rightarrow 5 \rightarrow 9 \rightarrow 11$. (c) Aggregated backup-path performance under random robot failures in 20 trials, with 5–9 robots (25%–45%) failing per trial. For each failure rate, the mean backup-path hop count (of all surviving follower robots, all 20 trials) according to primary-path hop count is shown, with one-sigma error bars (no interpolation).

For the reconfiguration process, we assume that the robot dynamics adhere to a single integrator model and utilize an illustrative distance-based formation control law as provided in(Oh et al., 2015). The reconfiguration follows a process where each surviving robot assesses its proximity to the desired formation configuration and assumes the role of the nearest removed robot, as dictated by the reconfiguration algorithm. For instance, robot 3 takes over the role of robot 2, robot 5 replaces robot 4, etc. The formation tracking error for each robot post-reconfiguration is depicted in Fig. 6.6 for the reconfiguration example given in Fig.6.5. The errors are presented as a function of time, showing the asymptotic convergence to zero for all robots, with varying rates depending on their hierarchical level. The tracking errors for robots at lower hierarchical levels diminish rapidly, while those at higher levels converge more slowly. This hierarchical convergence behavior ensures that the overall formation stability is prioritized, with the root and critical robots in the formation tree achieving stability first.

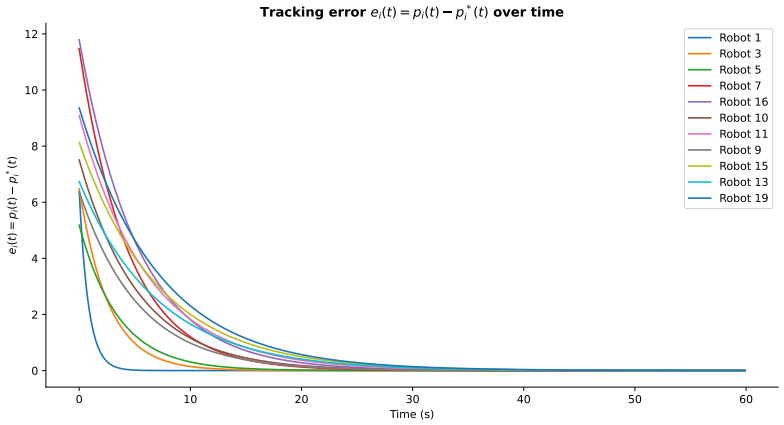


Figure 6.6: Formation tracking error for each robot following formation reconfiguration.

6.7.2 Example scenario 2: Intermittent faults

To evaluate detection and mitigation of intermittent faults, we use a setup like that in Sec. 6.7.1, but with a formation operating in 3D space while being subjected to

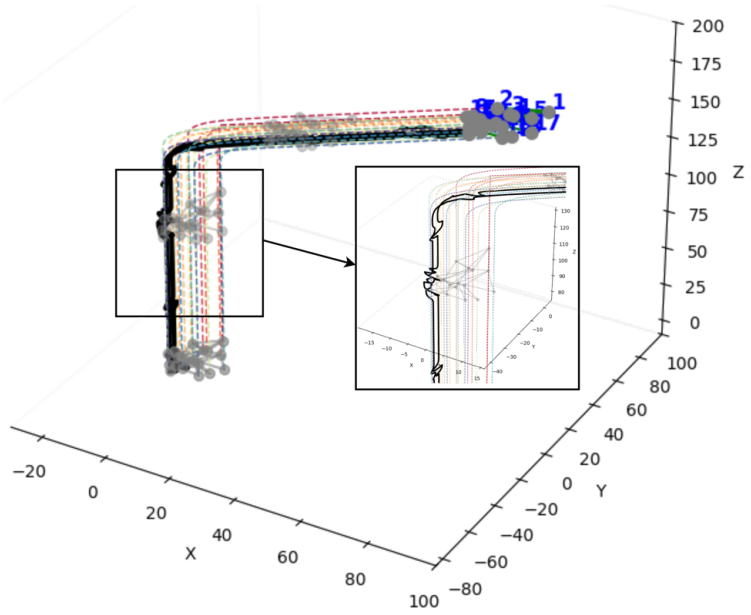
intermittent faults on the x and y axes. The goal is to maintain the formation while navigating, despite transient erroneous data.

The leader robot begins its motion along the z-axis at a constant speed. As the leader robot moves, the rest of the formation follows, maintaining relative positions as per the prescribed formation accompanying the HHC-constructed graph. During navigation, certain robots are exposed to intermittently faulty relative position data with the fault occurrence probability P_f, combined with communication noise modeled by the channel bit error probability P_e. In Fig. 6.7, the simulation result is presented for the robot at node 11 in the formation given in Fig. 6.3, as a representative example. The robot at node 11 is affected by faulty relative position data intermittently reported by one of its primary parents (robot at node 8). Initially, they follow a stable trajectory, moving along with the formation. Then, as intermittent faults affect the relative position data received from node 8, deviations are observed in their movement. These deviations, although minor initially, can accumulate over time, leading to noticeable discrepancies in the formation.

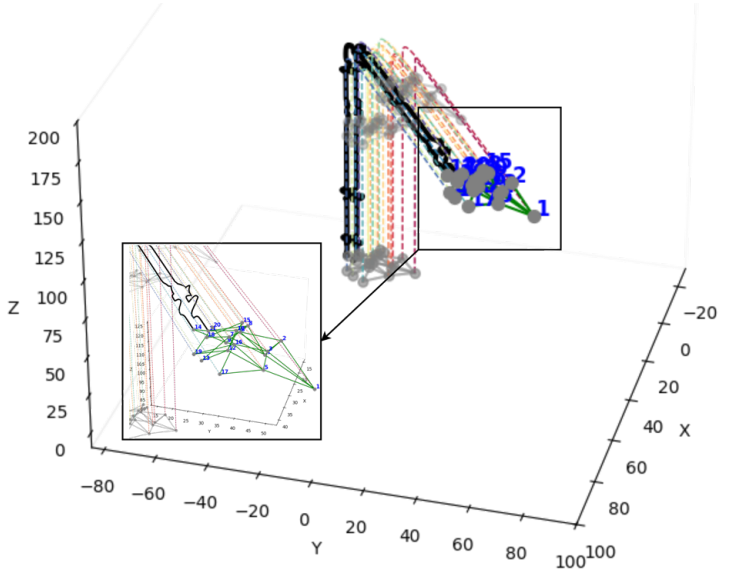
6.7.3 Detection of simple and complex intermittent faults

To assess the detection accuracy and false positives rate of our *proactive-reactive* method, we compare its performance to a benchmark case with a centralized detection network. In the centralized benchmark, the leader robot acts as an information hub: it disseminates the correct reference values to all robots, gathers all measurements returned by all robots (each corrupted only by channel bit error P_e), and then decides whether the sender is faulty or not. The samples the leader receives are randomly drawn from two overlapping distributions: f_0 if the sender is not faulty and f_1 (shifted by the fault offset) if the sender is faulty. To separate these noisy and partially overlapping distributions, the leader applies the *Bayes-optimal likelihood-ratio test* by computing $\Lambda = f_1/f_0$ and comparing this ratio with a fixed threshold that maximizes $\frac{1}{2}(\mathrm{TPR} + \mathrm{TNR})$ for the given noise level P_e.

In this section, we compare the performance of this centralized benchmark to that of our *proactive-reactive* method, in the formation control scenario of Sec. 6.7.2



(a)



(b)

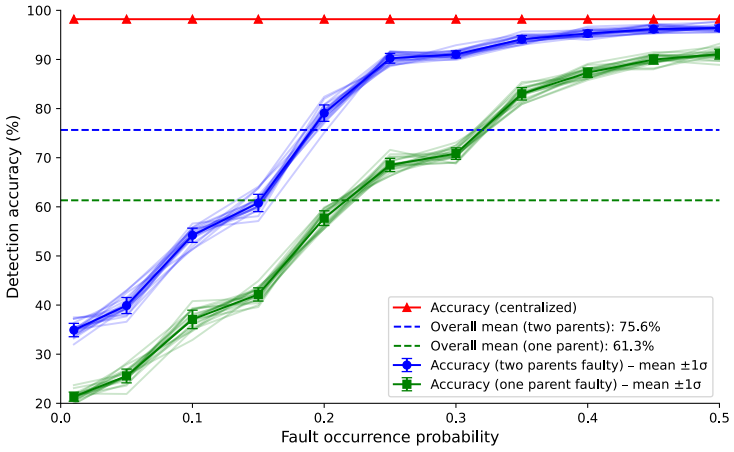
Figure 6.7: **Example scenario 2: The effect of intermittent faults without the use of a fault tolerance mechanism.** The effect of intermittent faults on a 3D navigation scenario, using the example formation given in Fig. 6.3. In this example, $P_f = 0.5$ and $P_e = 0.02$. Robots 11 and 14 are highlighted as illustrative examples, with their trajectories shown in bold black: robot 11 experiences intermittent offset faults in the relative-position data from its parent, robot 8. The inset highlights the moments when the error affects the trajectory of robots 11 and 14.

(see Fig. 6.7). We test our *proactive-reactive* method in the cases of one faulty parent and two faulty parents.

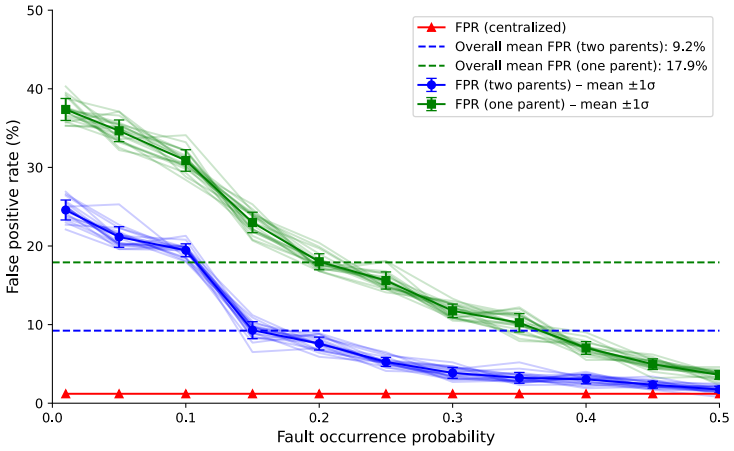
Fig. 6.8 presents the results of 20 independent Monte Carlo trials per fault probability P_f, with the channel bit error probability P_e held constant (at $P_e = 0.02$). In Fig. 6.8(a), the 1σ bands remain very narrow (under 3%) at low fault rates ($P_f \leq 0.10$), indicating highly consistent behavior across trials. Variability grows modestly at the mid fault rates ($P_f \approx 0.15$–0.25), especially for the case of one faulty parent. For the case of two faulty parents, only a few outlier runs remain before variability reaches nearly zero at the high fault rates ($P_f \geq 0.40$). For the case of one faulty parent, the accuracy rises gradually from $\approx 22\%$ to $\approx 90\%$. With two faulty parents, accuracy climbs from $\approx 34\%$ at $P_f = 0.01$ to over 90% by $P_f = 0.30$, approaching the centralized bound. Similarly to the accuracy, the false-positive rate variability (see Fig. 6.8(b)) reaches nearly 4% for the case of two faulty parents by $P_f = 0.35$.

Fig. 6.9 presents the mean detection accuracy (sensitivity) and false positive rate (specificity) of our *proactive-reactive* method, computed over 20 Monte Carlo runs per grid cell, for varying fault occurrence probabilities $P_f \in [0.01, 0.50]$ and channel bit error probabilities $P_e \in [0.01, 0.25]$. For the case of one faulty parent, Fig. 6.9(a) demonstrates that detection accuracy climbs steeply with P_f even under light noise. At $P_e = 0.01$, accuracy rises from about 27% at $P_f = 0.01$ to cross the 50% contour at $P_f = 0.20$, and the 75% contour once $P_f \gtrsim 0.35$. Accuracy only falls below 50% in the extreme low-P_f, high-P_e corner, and remains above 10% across most of the (P_f, P_e) plane. Fig. 6.9(c) shows false positives beginning at 33% for $(P_f = 0.01, P_e = 0.01)$, and falling below the 10% contour for $P_f \gtrsim 0.35$. The 60% accuracy and 10% false positives contours delineate a safe operating envelope of roughly $P_f \gtrsim 0.35, P_e \lesssim 0.10$, within which the sensitivity versus specificity of detection is balanced.

For the case of two faulty parents, Fig. 6.9(b) exhibits an even sharper rise in accuracy: under minimal noise ($P_e = 0.01$), accuracy rises from $\lesssim 40\%$ at $P_f = 0.01$ to cross the 50% accuracy contour by $P_f = 0.05$, and the 90% contour appears near $P_f = 0.20$. As P_e increases these breakpoints shift rightward. In Fig. 6.9(d),



(a)



(b)

Figure 6.8: **Detection of simple intermittent faults.** Performance of the fault detection strategy under varying fault-occurrence probabilities for the formation control scenario given in Fig. 6.7, at fixed channel bit error probability P_e0.02. (a) Detection accuracy: 20 independent simulation runs for each case (two faulty parents shown in blue, one faulty parent shown in green). Thin lines show individual trials; bold lines show the mean $\pm 1\sigma$ across runs, with vertical error bars. Dashed horizontal lines show the overall average accuracy across all fault probabilities for each case (centralized benchmark is shown in red). (b) False positive rate, in the same plot style as (a).

false-positives start at about 21% and fall under the 10% contour by $P_f = 0.15$ at $P_e = 0.01$. The 75% accuracy and 10% false positive contours delineate a safe envelope around $P_f \gtrsim 0.25, P_e \lesssim 0.10$, showing that even when both parents are faulty, detection sensitivity versus specificity can be balanced effectively.

Compared to the centralized benchmark (Fig. 6.9(e,f)), our *proactive-reactive* method sacrifices 10–30 % accuracy at very low fault rates or under heavy noise, yet it narrows the gap to within a few percentage points of the upper bound established by the centralized benchmark once moderate fault activity appears. It also achieves lower false positive rates compared to the centralized benchmark in the high-P_f, low-noise regimes that pose the greatest risk during bursts.

6.7.4 Demonstration in a swarm of 200 robots

This section presents a proof-of-concept demonstration of a 200-robot swarm under time-varying fault conditions, in the scenario of two faulty parents. We conducted 20 independent Monte Carlo simulation trials of a 200-robot HHC-constructed formation exposed to IFs, both with our *proactive-reactive* method and in a baseline swarm with no IF-tolerance strategy. In the simulation scenario, we first expose both swarms to a low background probability ($P_f = 0.1$) of IFs occurring randomly in robots at each second. Our *proactive-reactive* method performs almost perfectly during this period, while the baseline swarm experiences full breakdown. Therefore, to also study the limitations of our *proactive-reactive* method, we secondly expose both swarms to a high-intensity fault burst ($P_f = 0.35$) modeled as a fixed time interval ($t = 30$ to $t = 50$ seconds).

Fig. 6.10(a) presents the mean absolute tracking error $\bar{e}(t)$, averaged across all robots and trials. Without any fault tolerance or mitigation strategy (red), errors accumulate rapidly and exceed the breakdown threshold (dashed black line) well before the burst, after which they remain saturated. With our *proactive-reactive* method (green), error growth is well-contained during the background phase, but increases more rapidly during the burst interval. The increase during the burst can be attributed to the fact that, although per-fault detection accuracy improves as the

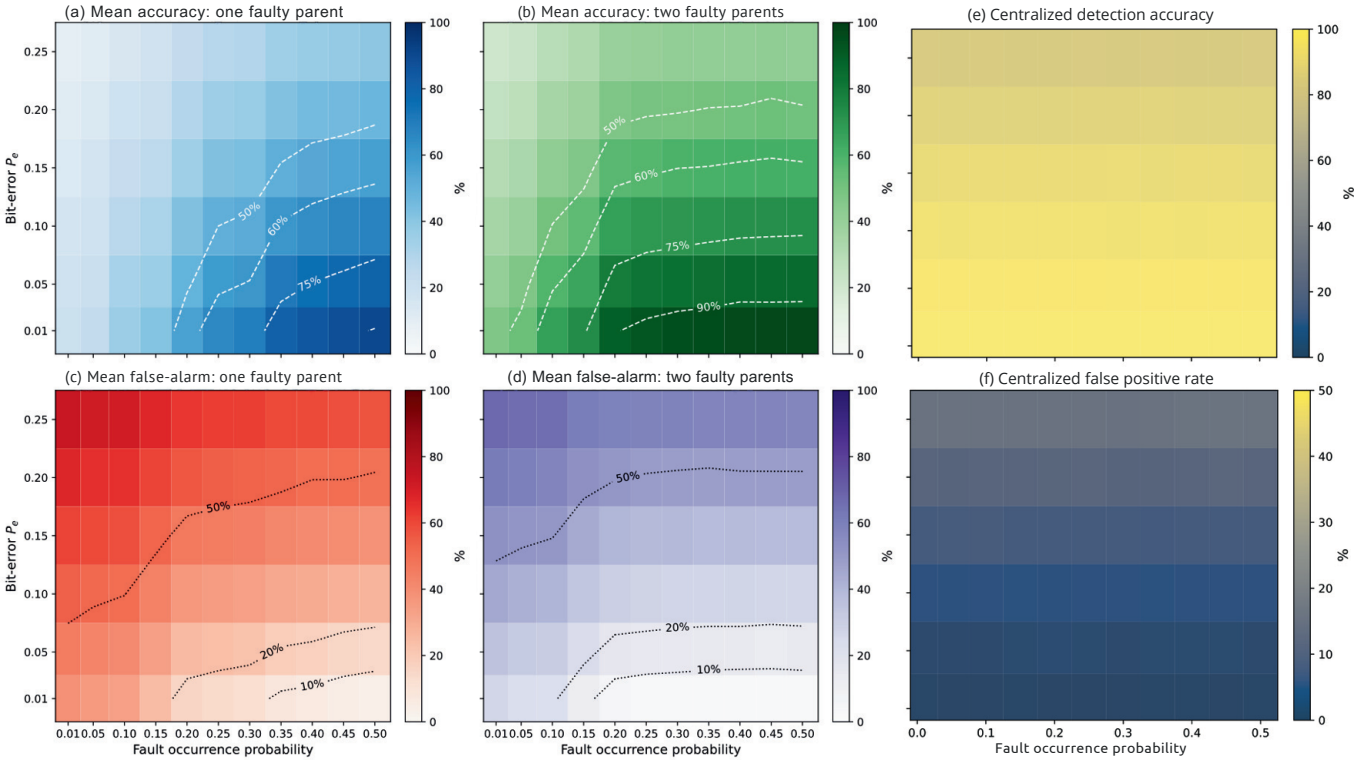


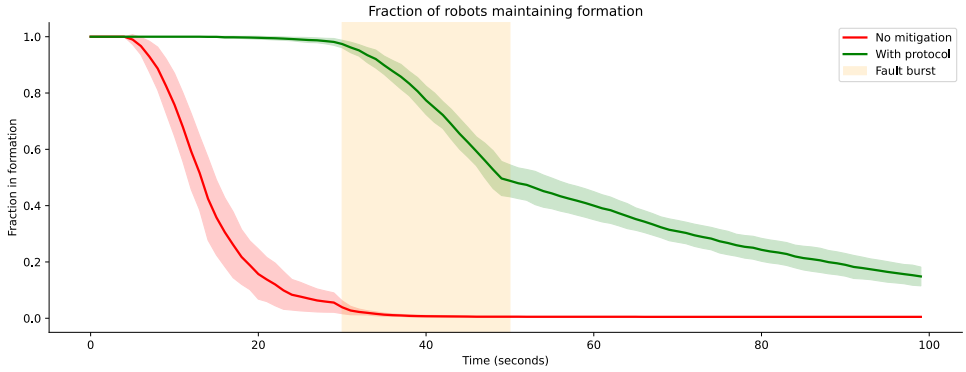
Figure 6.9: **Detection of complex intermittent faults.** (a-d) Detection performance of our *proactive-reactive* method under joint offset faults and channel noise, in the cases of (a,c) one faulty parent and (b,d) two faulty parents. Heatmaps (20 trials per grid-cell) showing the (a-b) mean detection accuracy and (c-d) false positives rate as functions of fault occurrence probability P_f and bit-error probability per hop P_e. (e) Mean detection accuracy, and (f) mean false positive rate of the centralized benchmark, as a function of the fault occurrence probability P_f, for varying channel bit-error probabilities P_e.

fault occurrence rate increases (see Fig. 6.9), the total fault occurrence is so high that the absolute rate of undetected faults can stay constant or rise (i.e., multiplying by a larger P_f can outweigh the accuracy gain). For example, moving from $P_f = 0.10$ at 60% detection to $P_f = 0.35$ at 90% detection changes only from $0.10 \times 0.40 = 0.04$ to $0.35 \times 0.10 = 0.035$. Because each undetected fault contributes permanently to cumulative error, the green curve in Fig. 6.10(a) still climbs during the burst—though it remains below the threshold far longer than in the unmitigated case. After the burst, error growth reverts to its slower background rate.

Fig. 6.10(b) shows the proportion of robots maintaining the formation with a tracking error below the breakdown threshold, over time. Without any fault tolerance or mitigation strategy (red), the formation fraction approaches zero (i.e., all robots exceeding the allowable error) within 25 s and remains there. With our *proactive-reactive* method (green), nearly all robots in the swarm hold the formation before the burst, and during the burst they experience a gradual decline rather than a catastrophic collapse. After the burst, the loss rate reduces substantially, but around half of the robots have already been lost and the swarm cannot recover its pre-burst rate (which was nearly no loss). Still, at the end of the trial at $t = 100$ s, a substantial core of robots (> 30) is still active in the formation.

Overall, the method is extremely effective at detecting and mitigating IFs in low background probability ($P_f = 0.1$) in a 200-robot swarm, especially compared to the baseline swarm with no IF-tolerance method, which degrades completely even under this low IF condition. During a high-intensity burst of IFs, although the per-fault detection accuracy improves during the burst, the much higher volume of faults results in the *absolute rate* of undetected faults still growing, so the total number of undetected events remains substantial. In short, in large swarms experiencing substantial high-intensity bursts of IFs, a fault saturation threshold does exist, after which *formation quality* will still degrade when using our *proactive-reactive* method, in its current form. However, under this fault saturation threshold, our *proactive-reactive* method is highly effective, dramatically outperforming the unmitigated baseline and enabling robust operation even in challenging conditions.

(a)



(b)

Figure 6.10: **Simulation results with a swarm of 200 robots.** Tracking performance of a 200-robot hierarchical swarm under time-varying IFs, with a burst period ($P_f = 0.35$) from $t = 30$ to $t = 50$ s and a low background fault rate ($P_f = 0.1$) otherwise. Results are averaged over 20 independent trials, with shaded regions indicating $\pm 1\sigma$ across trials. The orange band marks the fault burst interval. The impact of the burst phase is governed by the rate of undetected faults, which is determined by: *undetected fault rate* $= P_f \times (1 -$ detection accuracy$)$. (a) Mean absolute tracking error $\overline{e}(t)$ in meters. Tracking errors are cumulative, and robots that exceed the breakdown threshold (shown as dashed black line) are counted as irrecoverable for the remainder of the trial. (b) Proportion of robots maintaining the formation as a function of time.

6.8 Chapter conclusions

In this chapter, we presented a novel *proactive–reactive* fault-tolerance strategy designed to address the challenges posed by intermittent faults (IFs) in robot swarms, particularly for establishing reliable multi-hop communication paths and maintaining desired formations. Proactively, the strategy uses *adaptive biased minimum consensus* (ABMC) for robots to self-organize dynamic backup communication paths before faults occur. The ABMC protocol extends the existing biased minimum consensus (BMC) to dynamic networks of mobile robots. ABMC incorporates dynamic network characteristics, dynamic robot relative positions, and dynamic bias terms, and also allows the establishment of new edges not already present in the original network. This protocol ensures that each robot can continually adapt its backup path to the leader robot, enhancing the swarm's resilience to communication disruptions. Reactively, the strategy uses one-shot likelihood ratio tests for fault detection and mitigation. By comparing information received via primary and backup paths, robots can detect statistically significant deviations indicative of IFs. Upon IF detection along a primary path, communication is temporarily rerouted through a backup path, until the IF resolves. Thus, reliable flow of positional information is maintained and the effect of IFs on the swarm's formation control performance is minimized. Overall, this chapter provides a proof-of-concept of proactive–reactive fault tolerance against IFs in self-organized hierarchical robot swarms, a step towards addressing the broad challenges posed by IFs in general.

The present work delivers a proof-of-concept of detecting and mitigating data-centric IFs in a self-organized hierarchical robot swarm, but it would be important for future work to expand the types of data-centric IFs that are considered. Firstly, we have specifically addressed offset faults. While offset faults are indeed a common issue across various sensor modalities, actuator types, and communication channels, other data-centric intermittent faults, such as stuck-at and spike faults, also warrant attention. Secondly, the present work assumes that some fault-free path exists among the constructed multiplex network (primary paths and backup paths). However, in real-world scenarios, there could be IFs that affect all possible paths in the multiplex

network from a robot to its leader. Similarly, the present work assumes the leader itself is fault-free, but this will not always be the case in real-world scenarios. Future research could investigate fault tolerance methods for IFs that synchronously affect many robots (e.g., a large proportion of the robots with few hops to the leader, or a large proportion of the swarm overall) as well as IFs that effect the leader robot. Finally, future work should also validate the approach with larger swarms exposed to IFs in complex environments, as well as validate the approach with real robots exposed to IFs in field experiments.

Chapter 7

Conclusions and Discussions

This chapter summarizes the main contributions of the thesis, discusses the current limitations of the results, discusses the implications for aerial swarm robotics, and outlines potential directions for future research.

7.1 Summary of contributions

This thesis advances resilient coordination in aerial robot swarms by integrating self-organized hierarchy, reconfigurable network topologies, and proactive fault-tolerance mechanisms. It also bridges the gap between theory and practice through the introduction of the S-drone platform for experimental validation. Specifically, the thesis makes the following key contributions:

1. **Development of an open-source UAV platform (S-drone):** Chapter 3 introduced the S-drone, a modular and extensible UAV platform for swarm robotics research. Designed to support decentralized coordination and inter-robot tracking, the platform can operate without any external infrastructure, such as GPS or motion tracking. The S-drone platform thus facilitates experimental validation of swarm robotics algorithms, supporting tasks such as cooperative transport and navigation. This contribution provides a practical foundation for validating the resilient swarm behaviors and hierarchical frameworks presented in the following chapter.

2. **Self-organizing hierarchy through the Self-organizing Nervous System (SoNS):** Chapter 4 presented the SoNS framework to dynamically form self-organizing hierarchical structures for scalable, flexible, and fault-tolerant swarm coordination. Proof-of-concept missions demonstrated its ability in tasks such as search-and-rescue. Through hierarchical coordination and dynamic leadership, the SoNS enables a whole robot swarm to be programmed as if it were a single robot with a reconfigurable body. It enables mission-specific reconfigurations and allows coordinated decision-making without sacrificing the scalability or fault tolerance benefits of self-organization.

3. **Graph-theoretic formulation for hierarchical frameworks:** Chapter 5 extended bearing rigidity theory to self-organizing hierarchical frameworks, enabling distributed algorithms for construction and reconstruction. The proposed Hierarchical Henneberg Construction (HHC) preserves rigidity and hierarchy during framework merging, splitting, and reconfiguration, as well as the addition or loss of individual robots, establishing a mathematically grounded framework for resilient coordination in aerial robot swarms. The chapter also provided guarantees of structural stability during dynamic reconfigurations and permanent connection faults (e.g., losing a robot) when using HHC, thus establishing its scalability and resilience. The approach addresses both theoretical challenges and practical needs for scalable and flexible aerial swarm formations.

4. **Proactive–reactive fault tolerance strategy for intermittent faults (IFs):** Chapter 6 presented a decentralized approach to detect and mitigate IFs: transient disturbances that intermittently corrupt sensor or communication data. Such faults can destabilize swarm coordination before traditional reactive methods are able to respond, making them a critical challenge in aerial swarm deployments operating in dynamic and uncertain environments. The presented approach to address such faults included two key parts: proactive and reactive. Proactively, the Adaptive Biased Minimum Consensus (ABMC) protocol was introduced, which adaptively re-weights local consensus updates in order to establish and sustain backup multi-hop communication paths. In this

way, alternative communication paths are already in place before a fault occurs, reducing recovery delays. Reactively, a likelihood-ratio test (LRT) is applied to prediction errors that represent the difference between data a robot receives from different paths through the network. Once the presence of faulty information is detected by statistically monitoring the deviations, the affected links are down-weighted or excluded, and information flow is immediately rerouted through the proactively prepared backup paths. Overall, Chapter 6 has provided a practical and scalable proactive–reactive fault-tolerance approach that directly addresses a failure mode common in real-world aerial swarms, thereby enhancing their operational reliability in applications where continuous and dependable performance is essential. This contribution strengthens swarm resilience by mitigating transient failures, ensuring seamless rerouting around faulty robots and preserving mission continuity.

7.2 Limitations

To assess the applicability of the results to aerial swarm robotics in general, as well as to guide directions for future research, I identify the following main limitations of the current work:

- **Scalability in real-world deployments:** While simulations confirmed scalable performance for swarms containing up to 250 robots, practical demonstrations have been limited to approximately 20 units. This limitation was due primarily to practical constraints in laboratory infrastructure, including available physical space, battery-management logistics, and bandwidth saturation under heavy multi-agent communication loads. Such real-world bottlenecks pose challenges not fully captured in simulation, particularly congestion-induced communication failures or cascading fault events. Consequently, the lack of empirical validation with swarms of real robots at larger scales presents an important current limitation.

- **Outdoor operation:** The current S-drone platform utilizes sensor suites and

control algorithms designed specifically for indoor operations at relatively low speeds. Challenges related to outdoor operations, such as coping with external disturbances (e.g., wind gusts, weather variability), and higher-speed dynamic maneuvers (typical in realistic search-and-rescue scenarios at high speeds), have not yet been addressed.

- **Hierarchical reconfiguration in real-world deployments:** The hierarchical coordination schemes presented 5 provide formal guarantees of bearing rigidity, and 4 provides proof-of-concept demonstrations of self-organizing hierarchy with real robots as well as simulated experiments testing some temporary system-wide faults (sensor occlusion and communication outage). However, real-world performance in a variety of practical contexts remains untested. In practice, reconfiguration unfolds in network environments where communication links are imperfect—characterized by variable latency, intermittent packet loss, and fluctuating bandwidth. These factors can potentially slow down or destabilize coordination, which carries the risk of causing the intended hierarchy construction to lag behind the swarm's physical motion and, in extreme cases, fragment into disconnected sub-groups. Moreover, the current simplifying assumptions in 5—including undirected lower-triangular graphs, obstacle-free environments, and symmetric information flow—will not be able to be met in all practical deployments.

- **Fault-model coverage:** The current proactive-reactive fault-tolerance strategy presented in 6 targets intermittent faults (e.g., temporary sensor error) in some of the robots in a swarm and presumes that it is possible to construct backup paths with the remaining fault-free robots. Realistically, swarm deployments may also experience more severe fault scenarios, including persistent actuator or sensor faults ("stuck-at" conditions), as well as sudden system-wide spikes characterized by short-duration, high-intensity bursts of erroneous data or transient anomalies in data streams. In principle, multiple fault-tolerance mechanisms for different types of faults could be deployed in one system. However, the current IF fault-tolerance framework does not have a mechanism to

distinguish persistent or system-wide faults from more isolated transient errors, and thus critical failures could incorrectly be interpreted and responded to as noise.

7.3 Implications and applications

The theoretical and experimental advances presented in this thesis pave the way for more capable, resilient aerial robot swarms across a range of application domains. Below we highlight three representative areas and explain how our key contributions can translate into operational benefits:

- **Search-and-Rescue in unstructured environments:** Reliable coordination and adaptability make the methods presented in this thesis suitable for missions in dynamic and unpredictable disaster environments. Hierarchical frameworks enable distributed sensing, task allocation, and flexible reorganization in scenarios characterized by rapid state transitions, stringent latency constraints, and limited bandwidth, thereby enhancing robustness and improving the likelihood of mission success.
- **Large-scale environmental monitoring and mapping:** Scalable swarms can efficiently cover large areas for ecological surveys or resource mapping, especially in remote and hard-to-reach areas. Self-organizing hierarchy supports the division of tasks and dynamic adaptation to terrain constraints, ensuring effective coverage.
- **Persistent infrastructure inspection and monitoring:** Fault-tolerant formations support uninterrupted operation in industrial and urban settings, supporting long-duration cooperative monitoring of infrastructure. The fault-tolerance mechanisms developed in this thesis make hierarchical swarms reliable even in harsh environments where intermittent faults are likely, such as during inspection of bridges or of electrical and telecommunication infrastructure.

7.4 Future research directions

Despite the impact our contributions can already have on a range of application domains, several open challenges remain. In order to address the limitations identified in Section 7.2 and to extend the applicability of the results presented in this thesis, I propose the following directions for future investigation:

- **Scalable outdoor deployment and validation:** To address the scalability gap between simulation and real-world experiments, as well as extend to outdoor scenarios, the following will need to be integrated: lightweight LTE/5G ad-hoc mesh networks to support communication outdoors, automated battery management systems to facilitate rapid battery exchange or charging, and cloud-based data logging infrastructures for post-hoc performance evaluation. Empirical studies conducted in outdoor environments can expose potential operational constraints, for example due to communication congestion or cascading fault propagation, thereby enabling the refinement of scalable coordination strategies. Moreover, expanding the S-drone's capabilities to outdoor environments or dynamic operational contexts would require additional hardware and software modifications beyond those to support the already listed integrations, such as support for extended state estimation.

- **Generalization of hierarchical coordination and dynamic reconfiguration:** To expand hierarchical frameworks to scenarios with a wider range of practical constraints, future theoretical developments should address scenarios in which the assumptions of lower-triangular structures and symmetric communication links made in 5 will fail. It will also be important to integrate obstacle-aware navigation algorithms directly into the hierarchical reconfiguration process, enabling UAV formations to adapt their communication topology and spatial organization proactively as environmental conditions evolve.

- **Comprehensive fault detection, isolation, and recovery:** Future research should expand the fault detection and isolation algorithms to handle a wider range of fault scenarios, for example scenarios in which the possibility

of constructing backup communication paths via some proportion of fault-free robots cannot be assumed. Future research should also develop comprehensive recovery mechanisms capable of maintaining operational integrity in realistic multi-fault conditions (e.g., conditions that simultaneously include IFs in some proportion robots as well as system-wide spikes of erroneous data). Incorporating machine learning methods—such as autoencoder-based anomaly detectors for real-time sensor outlier rejection, graph neural networks for pinpointing faulty links or agents, and multi-agent reinforcement learning policies that reassign roles and rewire connectivity on the fly—is a promising future direction for adaptive fault management and rapid swarm reconfiguration after failures.

7.5 Concluding remarks

This thesis proposes a framework for resilient coordination in aerial robot swarms, integrating self-organizing hierarchy that was demonstrated with real robots using a custom-developed UAV platform, with reconfigurable network topologies, bearing-based coordination frameworks, and proactive–reactive fault tolerance. By addressing scalability, adaptability, and fault tolerance, the thesis advances the state of the art in aerial swarm robotics. The development of the S-drone platform also bridges theoretical methods with real-world implementation, providing a practical foundation for future research in aerial swarm robotics.

As a whole, the results presented in this thesis support distributed sensing, self-organized formation control, and robust swarm coordination suitable for a range of potential applications in dynamic and uncertain environments. The contributions of this thesis lay the groundwork for more autonomous, flexible, scalable but also manageable, robust, and trustworthy aerial robots. They provide practical tools and methodologies to inspire further innovations in swarm intelligence, swarm robotics, and multi-robot systems. Future research can build upon these foundations to address remaining challenges and unlock new possibilities for resilient and intelligent aerial swarm coordination.

Bibliography

M. Abdelkader, S. Güler, H. Jaleel, and J. S. Shamma. Aerial swarms: Recent applications and challenges. *Current Robotics Reports*, 2:309–320, 2021.

S. Abdelwahed, G. Karsai, N. Mahadevan, and S. C. Ofsthun. Practical implementation of diagnosis systems using timed failure propagation graph models. *IEEE Transactions on Instrumentation and Measurement*, 58(2):240–247, 2008.

A. Ahmad, V. Walter, P. Petráček, M. Petrlík, T. Báča, D. Žaitlík, and M. Saska. Autonomous aerial swarming in GNSS-denied environments with high obstacle density. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 570–576. IEEE, 2021.

M. M. Alam and S. Moh. Joint topology control and routing in a UAV swarm for crowd surveillance. *Journal of Network and Computer Applications*, 204:103427, 2022.

N. Aliane. A survey of open-source UAV autopilots. *Electronics*, 13(23):4785, 2024.

S. A. Amraii, P. Walker, M. Lewis, N. Chakraborty, and K. Sycara. Explicit vs. tacit leadership in influencing the behavior of swarms. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2209–2214. IEEE, 2014.

G. Ariante and G. Del Core. Unmanned aircraft systems (UASs): Current state, emerging technologies, and future trends. *Drones*, 9(1):59, 2025.

T. Baca, M. Petrlik, M. Vrba, V. Spurny, R. Penicka, D. Hert, and M. Saska. The MRS UAV system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles. *Journal of Intelligent and Robotic Systems*, 102(1):1–28, 2021.

A. A. Baktayan, A. T. Zahary, A. Sikora, and D. Welte. Computational offloading into UAV swarm networks: a systematic literature review. *EURASIP Journal on Wireless Communications and Networking*, 2024(1):69, 2024.

B. Balázs, G. Vásárhelyi, and T. Vicsek. Adaptive leadership overcomes persistence–responsivity trade-off in flocking. *Journal of the Royal Society Interface*, 17(167): 20190853, 2020.

L. Bayindir and E. Şahin. A review of studies in swarm robotics. *Turkish Journal of Electrical Engineering and Computer Sciences*, 15(2):115–147, 2007.

P. Beigi, M. S. Rajabi, and S. Aghakhani. An overview of drone energy consumption factors and models. *Handbook of Smart Energy Systems*, pages 1–20, 2022.

I. Bekmezci, O. K. Sahingoz, and Ş. Temel. Flying ad-hoc networks (fanets): A survey. *Ad Hoc Networks*, 11(3):1254–1270, 2013.

R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

G. Beni. The concept of cellular robotic system. In *Proceedings of the 1988 IEEE International Symposium on Intelligent Control*, pages 57–58. IEEE Computer Society, 1988.

C. Bilaloğlu, M. Şahin, F. Arvin, E. Şahin, and A. E. Turgut. A novel time-of-flight range and bearing sensor system for micro air vehicle swarms. In *International Conference on Swarm Intelligence*, pages 248–256. Springer, 2022.

J. D. Bjerknes and A. F. Winfield. On fault tolerance and scalability of swarm robotic systems. In *Distributed Autonomous Robotic Systems: The 10th International Symposium*, pages 431–444. Springer, 2013.

E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: From natural to artificial systems*. Oxford University Press, 1999.

S. Bouabdallah and R. Siegwart. Full control of a quadrotor. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 153–158. IEEE, 2007.

A. Boukerche, B. Turgut, N. Aydin, M. Z. Ahmad, L. Bölöni, and D. Turgut. Routing protocols in ad hoc networks: A survey. *Computer networks*, 55(13):3032–3080, 2011.

M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7:1–41, 2013.

K. Breitfelder and D. Messina. IEEE 100: the authoritative dictionary of IEEE standards terms. *Standards Information Network IEEE Press. v879*, pages 1–11, 2000.

Y. Bu, Y. Yan, and Y. Yang. Advancement challenges in UAV swarm formation control: A comprehensive review. *Drones*, 8(7):320, 2024.

F. Bullo, J. Cortés, F. Dörfler, and S. Martínez. *Lectures on network systems*, volume 1. CreateSpace, 2018.

M. Campion, P. Ranganathan, and S. Faruque. UAV swarm communication and control architectures: A review. *Journal of Unmanned Vehicle Systems*, 7(2): 93–106, 2018.

D. Carboni, R. K. Williams, A. Gasparri, G. Ulivi, and G. S. Sukhatme. Rigidity-preserving team partitions in multiagent networks. *IEEE Transactions on Cybernetics*, 45(12):2640–2653, 2014.

L. K. Carvalho, M. V. Moreira, and J. C. Basilio. Diagnosability of intermittent sensor faults in discrete event systems. *Automatica*, 79:315–325, 2017.

C. Chen, B. Wang, C. X. Lu, N. Trigoni, and A. Markham. Deep learning for visual localization and mapping: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(12):17000–17020, 2023.

A. L. Christensen, R. O'Grady, and M. Dorigo. From fireflies to fault tolerant swarms of robots. *IEEE Transactions on Evolutionary Computation*, 13(4):754–766, 2009.

S.-J. Chung, A. A. Paranjape, P. Dames, S. Shen, and V. Kumar. A survey on aerial swarm robotics. *IEEE Transactions on Robotics*, 34(4):837–855, 2018.

F. H. Clarke. *Optimization and nonsmooth analysis*. SIAM, 1990.

T. Clausen and P. Jacquet. Optimized link state routing protocol (olsr). Technical report, 2003.

M. Coppola, K. N. McGuire, C. De Wagter, and G. C. de Croon. A survey on swarming with micro air vehicles: Fundamental challenges and constraints. *Frontiers in Robotics and AI*, 7:18, 2020.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. The Ford-Fulkerson method. *Introduction to Algorithms*, pages 651–664, 2001.

J. Cortés. Distributed algorithms for reaching consensus on general functions. *Automatica*, 44(3):726–737, 2008.

F. Dalmao and E. Mordecki. Cucker–Smale flocking under hierarchical leadership and random interactions. *SIAM Journal on Applied Mathematics*, 71(4):1307–1316, 2011.

A. K. Das, R. Fierro, V. Kumar, J. P. Ostrowski, J. Spletzer, and C. J. Taylor. A vision-based formation control framework. *IEEE Transactions on Robotics and Automation*, 18(5):813–825, 2002.

M. C. De Gennaro and A. Jadbabaie. Decentralized control of connectivity for multi-agent systems. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 3628–3633. IEEE, 2006.

M. De Queiroz, X. Cai, and M. Feemster. *Formation control of multi-agent systems: a graph rigidity approach*. John Wiley & Sons, 2019.

P. G. F. Dias, M. C. Silva, G. P. Rocha Filho, P. A. Vargas, L. P. Cota, and G. Pessin. Swarm robotics: A perspective on the latest reviewed concepts and applications. *Sensors*, 21(6):2062, 2021.

J. Doornbos, K. E. Bennin, Ö. Babur, and J. Valente. Drone technologies: A tertiary systematic literature review on a decade of improvements. *IEEE Access*, 12:23220–23239, 2024.

M. Dorigo. Swarm-bot: An experiment in swarm robotics. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pages 192–200. IEEE, 2005.

M. Dorigo, V. Trianni, E. Şahin, R. Groß, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, and L. M. Gambardella. Evolving self-organizing behaviors for a swarm-bot. *Autonomous Robots*, 17(2–3):223–245, 2004.

M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. L. Christensen, A. Decugnière, G. Di Caro, F. Ducatelle, E. Ferrante, A. Förster, J. Guzzi, V. Longchamp, S. Magnenat, J. Martinez Gonzales, N. Mathews, M. Montes de Oca, R. O'Grady, C. Pinciroli, G. Pini, P. Rétornaz, J. Roberts, V. Sperati, T. Stirling, A. Stranieri, T. Stützle, V. Trianni, E. Tuci, A. E. Turgut, and F. Vaussard. Swarmanoid: A novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics and Automation Magazine*, 20(4):60–71, 2013.

M. Dorigo, M. Birattari, and M. Brambilla. Swarm robotics. *Scholarpedia*, 9(1): 1463, 2014.

M. Dorigo, G. Theraulaz, and V. Trianni. Reflections on the future of swarm robotics. *Science Robotics*, 5(49):eabe4385, 2020.

M. Dorigo, G. Theraulaz, and V. Trianni. Swarm robotics: Past, present, and future [point of view]. *Proceedings of the IEEE*, 109(7):1152–1165, 2021.

F. Ducatelle, A. Förster, G. A. Di Caro, and L. M. Gambardella. New task allocation methods for robotic swarms. In *Proceedings of the 9th IEEE/RAS Conference on Autonomous Robot Systems and Competitions (ICARSC)*, volume 5. IPCB: Instituto Politécnico de Castelo Branco, 2009.

F. Ducatelle, G. A. Di Caro, C. Pinciroli, and L. M. Gambardella. Self-organized cooperation between robotic swarms. *Swarm Intelligence*, 5:73–96, 2011.

J. Farrell. *Aided navigation: GPS with high rate sensors*. McGraw-Hill, Inc., 2008.

Z. Firat, E. Ferrante, Y. Gillet, and E. Tuci. On self-organised aggregation dynamics in swarms of robots with informed robots. *Neural Computing and Applications*, 32 (17):13825–13841, 2020.

A. Fishberg and J. P. How. Multi-agent relative pose estimation with uwb and constrained communications. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 778–785. IEEE, 2022.

D. Floreano and C. Mattiussi. *Bio-inspired artificial intelligence: Theories, methods, and technologies*. MIT press, 2008.

P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio, et al. Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight. *Science Robotics*, 7(67):eabl6259, 2022.

T. I. Fossen. *Guidance and control of ocean vehicles*. John Wiley & Sons, 1994.

G. K. Fourlas and G. C. Karras. A survey on fault diagnosis and fault-tolerant control methods for unmanned aerial vehicles. *Machines*, 9(9):197, 2021.

C. Ghedini, C. Ribeiro, and L. Sabattini. Toward fault-tolerant multi-robot networks. *Networks*, 70(4):388–400, 2017.

W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, and P. Kozierski. Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering. In *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 37–42. IEEE, 2017.

R. Groß, S. Nouyan, M. Bonani, F. Mondada, and M. Dorigo. Division of labour in self-organised groups. In *From Animals to Animats 10: 10th International Conference on Simulation of Adaptive Behavior, SAB 2008, Osaka, Japan, July 7-12, 2008. Proceedings 10*, pages 426–436. Springer, 2008.

D. Gu and Z. Wang. Leader–follower flocking: Algorithms and experiments. *IEEE Transactions on Control Systems Technology*, 17(5):1211–1219, 2009.

A. Guillen-Perez and M.-D. Cano. Flying ad hoc networks: A new domain for network communications. *Sensors*, 18(10):3571, 2018.

A. Guillen-Perez, A.-M. Montoya, J.-C. Sanchez-Aarnoutse, and M.-D. Cano. A comparative performance evaluation of routing protocols for flying ad-hoc networks in real conditions. *Applied Sciences*, 11(10):4363, 2021.

Z. J. Haas. A new routing protocol for the reconfigurable wireless networks. In *Proceedings of ICUPC 97-6th International Conference on Universal Personal Communications*, volume 2, pages 562–566. IEEE, 1997.

J. Halloy, G. Sempo, G. Caprari, C. Rivault, M. Asadpour, F. Tâche, I. Saïd, V. Durier, S. Canonge, J. M. Amé, C. Detrain, N. Correll, A. Martinoli, F. Mondada, R. Siegwart, and J.-L. Deneubourg. Social integration of robots into groups of cockroaches to control self-organized choices. *Science*, 318(5853):1155–1158, 2007.

H. Hamann. *Space–time continuous models of swarm robotic systems: Supporting global-to-local programming*. Springer, 2010.

H. Hamann. *Swarm robotics: A formal approach*. Springer, 2018.

H. Hamann and H. Wörn. A framework of space–time continuous models for algorithm design in swarm robotics. *Swarm Intelligence*, 2:209–239, 2008.

M. K. Heinrich, M. D. Soorati, T. K. Kaiser, M. Wahby, and H. Hamann. Swarm robotics: Robustness, scalability, and self-x features in industrial applications. *IT-Information Technology*, 61(4):159–167, 2019.

M. K. Heinrich, M. Wahby, M. Dorigo, and H. Hamann. Swarm robotics. In A. Cangelosi and M. Asada, editors, *Cognitive robotics*, pages 77–98. MIT Press, 2022.

J. M. Hendrickx, B. D. Anderson, J.-C. Delvenne, and V. D. Blondel. Directed graphs for the analysis of rigidity and persistence in autonomous agent systems. *International Journal of Robust Nonlinear Control*, 17(10-11):960–981, 2007.

D. Hert, T. Baca, P. Petracek, V. Kratky, R. Penicka, V. Spurny, M. Petrlik, M. Vrba, D. Zaitlik, P. Stoudek, et al. MRS drone: A modular platform for real-world deployment of aerial multi-robot systems. *Journal of Intelligent & Robotic Systems*, 108(4):64, 2023.

J. P. Hespanha, D. Liberzon, and A. S. Morse. Hysteresis-based switching algorithms for supervisory control of uncertain systems. *Automatica*, 39(2):263–272, 2003.

Y. Hou and C. Yu. Elementary operations for rigidity restoration and persistence analysis of multi-agent system. *IET Control Theory & Applications*, 10(2):119–125, 2016.

A. Howard. Multi-robot simultaneous localization and mapping using particle filters. *The International Journal of Robotics Research*, 25(12):1243–1256, 2006.

J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin. Fault-tolerant cooperative navigation of networked UAV swarms for forest fire monitoring. *Aerospace Science and Technology*, 123:107494, 2022.

M.-D. Hua, T. Hamel, P. Morin, and C. Samson. Introduction to feedback control of underactuated VTOL vehicles: A review of basic control design ideas and principles. *IEEE Control Systems Magazine*, 33(1):61–75, 2013.

X. Huang and D. Huang. Performance analysis of blockchain consensus algorithm in unmanned aerial vehicle ad hoc networks. *Drones*, 9(5):334, 2025.

A. Isidori. *Nonlinear control systems: An introduction*. Springer, 1985.

A. Isidori. *Nonlinear control systems II*. Springer, 2013.

A. Jamshidpey, W. Zhu, M. Wahby, M. Allwright, M. K. Heinrich, and M. Dorigo. Multi-robot coverage using self-organized networks for central coordination. In *Swarm Intelligence – Proceedings of ANTS 2020 – 12th International Conference*, pages 306–314, Berlin, Germany, 2020. Springer.

A. Jamshidpey, M. Dorigo, and M. K. Heinrich. Reducing uncertainty in collective perception using self-organizing hierarchy. *Intelligent Computing*, 2:0044, 2023.

A. Jamshidpey, M. Wahby, M. Allwright, W. Zhu, M. Dorigo, and M. K. Heinrich. Centralization vs. decentralization in multi-robot sweep coverage with ground robots and UAVs. *Artificial Life and Robotics*, 2025. DOI: 10.1007/s10015-025-01049-7.

Y. Jia and T. Vicsek. Modelling hierarchical flocking. *New Journal of Physics*, 21 (9):093048, 2019.

C. Ju and H. I. Son. A hybrid systems-based hierarchical control architecture for heterogeneous field robot teams. *IEEE Transactions on Cybernetics*, 53(3):1802–1815, 2021.

T. K. Kaiser and H. Hamann. Innate motivation for robot swarms by minimizing surprise: From simple simulations to real-world experiments. *IEEE Transactions on Robotics*, 38(6):3582–3601, 2022.

A. Khadidos, R. M. Crowder, and P. H. Chappell. Exogenous fault detection and recovery for swarm robotics. *IFAC-PapersOnLine*, 48(3):2405–2410, 2015.

B. Khaldi, F. Harrou, F. Cherif, and Y. Sun. Monitoring a robot swarm using a data-driven fault detection approach. *Robotics and Autonomous Systems*, 97:193–203, 2017.

H. K. Khalil and J. W. Grizzle. *Nonlinear systems*, volume 3. Prentice hall Upper Saddle River, NJ, 2002.

C. Kirst, M. Timme, and D. Battaglia. Dynamic information routing in complex networks. *Nature Communications*, 7(1):11061, 2016.

T. J. Koo and S. Sastry. Output tracking control design of a helicopter model based on approximate linearization. In *Proceedings of the 37th IEEE Conference on Decision and Control*, volume 4, pages 3635–3640. IEEE Press, 1998.

L. Krick, M. E. Broucke, and B. A. Francis. Stabilisation of infinitesimally rigid formations of multi-robot networks. *International Journal of Control*, 82(3):423–439, 2009.

G. Laman. On graphs and rigidity of plane skeletal structures. *Journal of Engineering mathematics*, 4(4):331–340, 1970.

G. Li, D. St-Onge, C. Pinciroli, A. Gasparri, E. Garone, and G. Beltrame. Decentralized progressive shape formation with robot swarms. *Autonomous Robots*, 43 (6):1505–1521, 2019.

X. Liang, Y.-H. Liu, H. Wang, W. Chen, K. Xing, and T. Liu. Leader-following formation tracking control of mobile robots without direct position measurements. *IEEE Transactions on Automatic Control*, 61(12):4131–4137, 2016.

H. Lin and P. J. Antsaklis. Stability and stabilizability of switched linear systems: a survey of recent results. *IEEE Transactions on Automatic control*, 54(2):308–322, 2009.

M. K. Marina and S. R. Das. On-demand multipath distance vector routing in ad hoc networks. In *Proceedings ninth international conference on network protocols. ICNP 2001*, pages 14–23. IEEE, 2001.

N. Mathews, A. L. Christensen, R. O'Grady, F. Mondada, and M. Dorigo. Mergeable nervous systems for robots. *Nature Communications*, 8(1):1–7, 2017.

L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Pixhawk: A system for autonomous flight using onboard computer vision. In *2011 IEEE International Conference on Robotics and Automation*, pages 2992–2997. IEEE, 2011.

G. Michieletto, A. Cenedese, and A. Franchi. Bearing rigidity theory in se (3). In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 5950–5955. IEEE, 2016.

A. Millard. *Exogenous fault detection in swarm robotic systems*. PhD thesis, University of York, 2016.

A. G. Millard, J. Timmis, and A. F. Winfield. Towards exogenous fault detection in swarm robotic systems. In *Towards Autonomous Robotic Systems: 14th Annual Conference, TAROS 2013, Oxford, UK, August 28–30, 2013, Revised Selected Papers 14*, pages 429–430. Springer, 2014.

A. G. Millard, R. Joyce, J. A. Hilder, C. Fleşeriu, L. Newbrook, W. Li, L. J. McDaid, and D. M. Halliday. The Pi-puck extension board: A Raspberry Pi interface for the e-puck robot platform. In *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 741–748. IEEE, 2017.

Y. Mo, L. Yu, and C. Yu. Global asymptotic stability of a general biased min-consensus protocol. *IET Control Theory & Applications*, 15(8):1148–1156, 2021.

K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, et al. Fast, autonomous flight in GPS-denied and cluttered environments. *Journal of Field Robotics*, 35(1):101–120, 2018.

F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th IEEE/RAS Conference*

on Autonomous Robot Systems and Competitions (ICARSC), pages 59–65. IPCB: Instituto Politécnico de Castelo Branco, 2009.

E. Montijano, E. Cristofalo, M. Schwager, and C. Sagues. Distributed formation control of non-holonomic robots without a global reference frame. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5248–5254. IEEE, 2016.

L. Moreau. Stability of multiagent systems with time-dependent communication links. *IEEE Transactions on Automatic Control*, 50(2):169–182, 2005.

P. T. Morón, S. Salimi, J. P. Queralta, and T. Westerlund. UWB role allocation with distributed ledger technologies for scalable relative localization in multi-robot systems. In *2022 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, pages 1–8. IEEE, 2022.

T. Muhammed and R. A. Shaikh. An analysis of fault detection strategies in wireless sensor networks. *Journal of Network and Computer Applications*, 78:267–287, 2017.

M. Nagy, Z. Ákos, D. Biro, and T. Vicsek. Hierarchical group dynamics in pigeon flocks. *Nature*, 464(7290):890–893, 2010.

N. P. Nguyen and P. Pitakwachara. Integral terminal sliding mode fault tolerant control of quadcopter UAV systems. *Scientific Reports*, 14(1):10786, 2024.

Y. Niu, L. Sheng, M. Gao, and D. Zhou. Distributed intermittent fault detection for linear stochastic systems over sensor network. *IEEE Transactions on Cybernetics*, 52(9):9208–9218, 2021.

S. Nouyan, R. Groß, M. Bonani, F. Mondada, and M. Dorigo. Teamwork in self-organized robot colonies. *IEEE Transactions on Evolutionary Computation*, 13 (4):695–711, 2009.

S. Oğuz, M. K. Heinrich, M. Allwright, W. Zhu, M. Wahby, E. Garone, and M. Dorigo. An open-source UAV platform for swarm robotics research: Using cooperative sensor fusion for inter-robot tracking. *IEEE Access*, 12:43378–43395, 2024.

K.-K. Oh and H.-S. Ahn. Formation control of mobile agents based on inter-agent distance dynamics. *Automatica*, 47(10):2306–2312, 2011.

K.-K. Oh, M.-C. Park, and H.-S. Ahn. A survey of multi-agent formation control. *Automatica*, 53:424–440, 2015.

J. O'Keeffe. Anticipating degradation: A predictive approach to fault tolerance in robot swarms. *arXiv preprint arXiv:2504.01594*, 2025.

J. O'Keeffe and A. G. Millard. Predictive fault tolerance for autonomous robot swarms. *arXiv preprint arXiv:2309.09309*, 2023.

O. Oladiran. *Fault Recovery in Swarm Robotics Systems using Learning Algorithms*. PhD thesis, University of York, 2019.

R. Olfati-Saber and R. Murray. Graph rigidity and distributed formation stabilization of multi-vehicle systems. In *Proceedings of the 41st IEEE Conference on Decision and Control*, pages 2965–2971, Las Vegas, NV, USA, Dec. 2002.

R. Olfati-Saber and R. M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on automatic control*, 49 (9):1520–1533, 2004.

R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

E. Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE International Conference on Robotics and Automation*, pages 3400–3407. IEEE, 2011.

O. S. Oubbati, M. Atiquzzaman, P. Lorenz, M. H. Tareque, and M. S. Hossain. Routing in flying ad hoc networks: Survey, constraints, and future challenge perspectives. *IEEE access*, 7:81057–81105, 2019.

S. Oğuz, M. K. Heinrich, M. Allwright, W. Zhu, M. Wahby, M. Dorigo, and E. Garone. A novel aerial robotic swarm hardware. In *Proceedings of the 41st Benelux Meeting on Systems and Control: Book of Abstracts*, page 96, Brussels, Belgium, 2022.

Y. Peng, H. Yang, Y. Cheng, and B. Jiang. Largest recoverable component based fault recoverability of UAV swarm with removal of faulty individuals. *Aerospace Science and Technology*, 118:107059, 2021.

C. Perkins, E. Belding-Royer, and S. Das. Rfc3561: Ad hoc on-demand distance vector (aodv) routing, 2003.

C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *ACM SIGCOMM computer communication review*, 24(4):234–244, 1994.

K. H. Petersen, N. Napp, R. Stuart-Smith, D. Rus, and M. Kovac. A review of collective robotic construction. *Science Robotics*, 4(28):eaau8479, 2019.

C. Pignotti and I. R. Vallejo. Flocking estimates for the Cucker–Smale model with time lag and hierarchical leadership. *Journal of Mathematical Analysis and Applications*, 464(2):1313–1332, 2018.

C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo. ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, 2012.

G. Pini, A. Brutschy, M. Birattari, and M. Dorigo. Task partitioning in swarms of robots: reducing performance losses due to interference at shared resources. In *Informatics in Control Automation and Robotics: Revised and Selected Papers from*

the International Conference on Informatics in Control Automation and Robotics 2009, pages 217–228. Springer, 2011.

W. Power, M. Pavlovski, D. Saranovic, I. Stojkovic, and Z. Obradovic. Autonomous navigation for drone swarms in gps-denied environments using structured learning. In *Artificial Intelligence Applications and Innovations: 16th IFIP WG 12.5 International Conference, AIAI 2020, Neos Marmaras, Greece, June 5–7, 2020, Proceedings, Part II 16*, pages 219–231. Springer, 2020.

H. N. Psaraftis, M. Wen, and C. A. Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31, 2016.

W. Ren and R. W. Beard. Consensus seeking in multiagent systems under dynamically changing interaction topologies. *IEEE Transactions on automatic control*, 50(5):655–661, 2005.

W. Ren, R. W. Beard, and E. M. Atkins. A survey of consensus problems in multiagent coordination. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 1859–1864. IEEE, 2005.

Y. Rizk, M. Awad, and E. W. Tunstel. Cooperative heterogeneous multi-robot systems: A survey. *ACM Computing Surveys (CSUR)*, 52(2):1–31, 2019.

H. L. Royden and P. Fitzpatrick. *Real analysis*, volume 32. Macmillan New York, 1988.

M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

I. Sa, M. Kamel, M. Burri, M. Bloesch, R. Khanna, M. Popović, J. Nieto, and R. Siegwart. Build your own visual-inertial drone: A cost-effective and open-source autonomous drone. *IEEE Robotics and Automation Magazine*, 25(1):89–103, 2017.

E. Şahin. Swarm robotics: From sources of inspiration to domains of application. In *Swarm Robotics, SAB2004 International Workshop, Santa Monica, CA, USA*, pages 10–20. Springer, 2004.

P. Samar, M. R. Pearlman, and Z. J. Haas. Independent zone routing: an adaptive hybrid routing framework for ad hoc wireless networks. *IEEE/ACM Transactions On Networking*, 12(4):595–608, 2004.

F. Schiano, A. Franchi, D. Zelazo, and P. R. Giordano. A rigidity-based decentralized bearing formation controller for groups of quadrotor UAVs. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5099–5106. IEEE, 2016.

P. Seibert and R. Suarez. Global stabilization of nonlinear cascade systems. *Systems and Control Letters*, 14(4):347–352, 1990.

H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani. Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges. *IEEE Access*, 7:48572–48634, 2019.

Q. Shan and S. Mostaghim. Collective decision making in swarm robotics with distributed bayesian hypothesis testing. In *Swarm Intelligence – Proceedings of ANTS 2020 – 12th International Conference*, pages 55–67. Springer, 2020.

L. Sheng, S. Zhang, and M. Gao. Intermittent fault detection for linear discrete-time stochastic multi-agent systems. *Applied Mathematics and Computation*, 410: 126480, 2021.

G. Skorobogatov, C. Barrado, and E. Salamí. Multiple UAV systems: A survey. *Unmanned Systems*, 8(02):149–169, 2020.

E. D. Sontag et al. On the input-to-state stability property. *European Journal of Control*, 1(1):24–36, 1995.

V. Strobel, E. Castelló Ferrer, and M. Dorigo. Managing byzantine robots via blockchain technology in a swarm robotics collective decision making scenario. In M. Dastani, G. Sukthankar, E. André, and S. Koenig, editors, *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*,

AAMAS '18, pages 541–549, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems. doi: http://dl.acm.org/citation.cfm?id=3237383.3237464.

W. A. Syed, S. Perinpanayagam, M. Samie, and I. Jennions. A novel intermittent fault detection algorithm and health monitoring for electronic interconnections. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 6 (3):400–406, 2016.

M. A. Tahir, I. Mir, and T. U. Islam. A review of UAV platforms for autonomous applications: comprehensive analysis and future directions. *IEEE Access*, 11:52540–52554, 2023.

M. S. Talamali, A. Saha, J. A. Marshall, and A. Reina. When less is more: Robot swarms adapt better to changes with constrained communication. *Science Robotics*, 6(56):eabf1416, 2021.

Z. Tang, R. Cunha, T. Hamel, and C. Silvestre. Relaxed bearing rigidity and bearing formation control under persistence of excitation. *Automatica*, 141:110289, 2022.

H. G. Tanner, V. Kumar, and G. J. Pappas. The effect of feedback and feedforward on formation ISS. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, volume 4, pages 3448–3453. IEEE Press, 2002.

H. G. Tanner, G. J. Pappas, and V. Kumar. Leader-to-formation stability. *IEEE Transactions on Robotics and Automation*, 20(3):443–455, 2004.

D. Tarapore, A. L. Christensen, and J. Timmis. Generic, scalable and decentralized fault detection for robot swarms. *PloS one*, 12(8):e0182058, 2017.

D. Tarapore, J. Timmis, and A. L. Christensen. Fault detection in a swarm of physical robots based on behavioral outlier detection. *IEEE Transactions on Robotics*, 35 (6):1516–1522, 2019.

D. Tarapore, R. Groß, and K.-P. Zauner. Sparse robot swarms: Moving swarms to real-world applications. *Frontiers in Robotics and AI*, 7:83, 2020.

S. Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

M. H. Trinh and H.-S. Ahn. Finite-time bearing-based maneuver of acyclic leader-follower formations. *IEEE Control Systems Letters*, 6:1004–1009, 2021.

M. H. Trinh, S. Zhao, Z. Sun, D. Zelazo, B. D. Anderson, and H.-S. Ahn. Bearing-based formation control of a group of agents with leader-first follower structure. *IEEE Transactions on Automatic Control*, 64(2):598–613, 2018.

M. H. Trinh, Q. Van Tran, and H.-S. Ahn. Minimal and redundant bearing rigidity: Conditions and applications. *IEEE Transactions on Automatic Control*, 65(10): 4186–4200, 2019.

A. Valada, N. Radwan, and W. Burgard. Deep auxiliary learning for visual localization and odometry. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6939–6946. IEEE, 2018.

G. Valentini, H. Hamann, and M. Dorigo. Efficient decision-making in a self-organizing robot swarm: On the speed versus accuracy trade-off. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1305–1314, 2015.

G. Valentini, E. Ferrante, H. Hamann, and M. Dorigo. Collective decision with 100 kilobots: Speed versus accuracy in binary discrimination problems. *Autonomous Agents and Multi-agent Systems*, 30(3):553–580, 2016.

V. S. Varadharajan, D. St-Onge, B. Adams, and G. Beltrame. Swarm relays: Distributed self-healing ground-and-air connectivity chains. *IEEE Robotics and Automation Letters*, 5(4):5347–5354, 2020.

V. S. Varadharajan, K. Soma, S. Dyanatkar, P.-Y. Lajoie, and G. Beltrame. Hierarchies define the scalability of robot swarms. *arXiv preprint arXiv:2405.02417*, 2024.

G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek. Optimized flocking of autonomous drones in confined environments. *Science Robotics*, 3(20):eaat3536, 2018.

M. Wahby, M. K. Heinrich, D. N. Hofstadler, E. Neufeld, I. Kuksin, P. Zahadat, T. Schmickl, P. Ayres, and H. Hamann. Autonomously shaping natural climbing plants: a bio-hybrid approach. *Royal Society Open Science*, 5(10):180296, 2018.

P. Walker, S. A. Amraii, N. Chakraborty, M. Lewis, and K. Sycara. Human control of robot swarms with dynamic leaders. In *Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1108–1113. IEEE, 2014a.

P. Walker, S. A. Amraii, M. Lewis, N. Chakraborty, and K. Sycara. Control of swarms with multiple leader agents. In *Proceedings of the 2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3567–3572. IEEE, 2014b.

V. Walter, N. Staub, A. Franchi, and M. Saska. UVDAR system for visual relative localization with application to leader–follower formations of multirotor UAVs. *IEEE Robotics and Automation Letters*, 4(3):2637–2644, 2019.

V. Walter, M. Vrba, and M. Saska. On training datasets for machine learning-based visual relative localization of micro-scale UAVs. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10674–10680. IEEE, 2020.

J. Wang, Y. Tan, and I. Mareels. Robustness analysis of leader-follower consensus. *Journal of Systems Science and Complexity*, 22(2):186–206, 2009.

Z. Wang, J. Li, J. Li, and C. Liu. A decentralized decision-making algorithm of UAV swarm with information fusion strategy. *Expert Systems with Applications*, 237: 121444, 2024.

A. C. Watts, V. G. Ambrosia, and E. A. Hinkley. Unmanned aircraft systems in remote sensing and scientific research: Classification and considerations of use. *Remote Sensing*, 4(6):1671–1692, 2012.

K. D. Wayne. *Generalized maximum flow algorithms*. Cornell University, 1999.

J. Werfel, K. Petersen, and R. Nagpal. Designing collective behavior in a termite-inspired robot construction team. *Science*, 343(6172):754–758, 2014.

A. F. Winfield and J. Nembrini. Safety in numbers: fault-tolerance in robot swarms. *International Journal of Modelling, Identification and Control*, 1(1):30–37, 2006.

H. Xu, Y. Zhang, B. Zhou, L. Wang, X. Yao, G. Meng, and S. Shen. Omni-swarm: A decentralized omnidirectional visual–inertial–UWB state estimation system for aerial swarms. *IEEE Transactions on Robotics*, 38(6):3374–3394, 2022.

R. Yan, X. He, Z. Wang, and D. Zhou. Detection, isolation and diagnosability analysis of intermittent faults in stochastic systems. *International Journal of Control*, 91(2):480–494, 2018.

A. Yaramasu, Y. Cao, G. Liu, and B. Wu. Aircraft electric system intermittent arc fault detection and location. *IEEE Transactions on Aerospace and Electronic Systems*, 51(1):40–51, 2015.

M. M. Zavlanos, M. B. Egerstedt, and G. J. Pappas. Graph-theoretic connectivity control of mobile robot networks. *Proceedings of the IEEE*, 99(9):1525–1540, 2011.

J. Zhang, P. D. Christofides, X. He, Z. Wu, Z. Zhang, and D. Zhou. Event-triggered filtering and intermittent fault detection for time-varying systems with stochastic parameter uncertainty and sensor saturation. *International Journal of Robust and Nonlinear Control*, 28(16):4666–4680, 2018.

Q. Zhang and H. H. Liu. Robust cooperative formation control of fixed-wing unmanned aerial vehicles. *arXiv preprint arXiv:1905.01028*, 2019.

S. Zhang, L. Sheng, M. Gao, and D. Zhou. Intermittent fault detection for discrete-time linear stochastic systems with time delay. *IET Control Theory and Applications*, 14(3):511–518, 2020.

S. Zhang, L. Sheng, and M. Gao. Intermittent fault detection for delayed stochastic systems over sensor networks. *Journal of the Franklin Institute*, 358(13):6878–6896, 2021.

Y. Zhang and S. Li. Distributed biased min-consensus with applications to shortest path planning. *IEEE Transactions on Automatic Control*, 62(10):5429–5436, 2017.

Y. Zhang, X. Wang, S. Wang, and X. Tian. Distributed bearing-based formation control of unmanned aerial vehicle swarm via global orientation estimation. *Chin. J. Aeronaut.*, 35(1):44–58, 2022.

Y. Zhang, S. Oğuz, S. Wang, E. Garone, X. Wang, M. Dorigo, and M. K. Heinrich. Self-reconfigurable hierarchical frameworks for formation control of robot swarms. *IEEE Transactions on Cybernetics*, 54(1):87–100, 2023.

S. Zhao and D. Zelazo. Bearing-based formation stabilization with directed interaction topologies. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 6115–6120. IEEE, 2015.

S. Zhao and D. Zelazo. Bearing rigidity theory and its applications for control and estimation of network systems: Life beyond distance rigidity. *IEEE Control Systems Magazine*, 39(2):66–83, 2019.

S. Zhao, Z. Sun, D. Zelazo, M.-H. Trinh, and H.-S. Ahn. Laman graphs are generically bearing rigid in arbitrary dimensions. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 3356–3361. IEEE, 2017.

H. Zheng, J. Panerati, G. Beltrame, and A. Prorok. An adversarial approach to private flocking in mobile robot teams. *IEEE Robotics and Automation Letters*, 5 (2):1009–1016, 2020.

D. Zhou, Z. Wang, and M. Schwager. Agile coordination and assistive collision avoidance for quadrotor swarms using virtual structures. *IEEE Transactions on Robotics*, 34(4):916–923, 2018.

D. Zhou, Y. Zhao, Z. Wang, X. He, and M. Gao. Review on diagnosis techniques for intermittent faults in dynamic systems. *IEEE Transactions on Industrial Electronics*, 67(3):2337–2347, 2019.

X. Zhou, X. Wen, Z. Wang, Y. Gao, H. Li, Q. Wang, T. Yang, H. Lu, Y. Cao, C. Xu, et al. Swarm of micro flying robots in the wild. *Science Robotics*, 7(66):eabm5954, 2022.

W. Zhu, M. Allwright, M. K. Heinrich, S. Oğuz, A. L. Christensen, and M. Dorigo. Formation control of UAVs and mobile robots using self-organized communication topologies. In *Swarm Intelligence – Proceedings of ANTS 2020 – 12th International Conference*, pages 306–314. Springer, 2020.

W. Zhu, S. Oğuz, M. K. Heinrich, M. Allwright, M. Wahby, A. L. Christensen, E. Garone, and M. Dorigo. Self-organizing nervous systems for robot swarms. *Science Robotics*, 9(96):eadl5161, 2024.