ULB

ECOLE
POLYTECHNIQUE
DE BRUXELLES

Swarm Oracle: Towards Trustless Self-Organization in Robot Swarms using Blockchain Technologies

Thesis presented by Alexandre PACHECO
in fulfilment of the requirements of the
PhD Degree in Engineering Sciences and Technology
("Doctorat en Sciences de l'Ingénieur et Technologie")
Academic year 2025-2026

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

Composition of the jury

Mauro Birattari *(chair)*
Université Libre de Bruxelles, Brussels, Belgium

Marco Dorigo *(thesis supervisor)*
Université Libre de Bruxelles, Brussels, Belgium

Mary Katherine Heinrich *(secretary)*
Université Libre de Bruxelles, Brussels, Belgium

Andreagiovanni Reina
Universität Konstanz; Max Planck Institute of Animal Behavior, Konstanz, Germany

Volker Strobel
Université Libre de Bruxelles, Brussels, Belgium

Tomi Westerlund
University of Turku, Turku, Finland

Abstract

The integration of blockchain technology with robotics offers a promising foundation for coordinating and securing decentralized multi-robot systems in real-world deployments. By maintaining a distributed record and a shared execution state, blockchains enable robots to enforce collective rules, synchronize data, log events, and detect inconsistencies in peer behavior without relying on centralized control.

A fundamental limitation, however, hinders the practical deployment of blockchain-enabled swarms: the oracle problem. Blockchain consensus follows a trustless design in which all participants independently verify and execute state transitions to maintain a consistent global state. While this design allows blockchains to operate in open environments with untrusted participants, it inherently restricts them to deterministic computations. In contrast, robotic systems rely on local perception derived from noisy and uncertain sensor data, making it difficult to directly integrate such information into blockchain-enabled decision processes. Blockchain oracles supply external data to the system, but many designs rely on trusted parties or centralized components, undermining decentralization and fault-tolerance.

This thesis introduces Swarm Oracle, a distributed oracle network composed of mobile robots that collectively validate uncertain environmental observations without assuming trust or cooperation among robots. Each robot submits locally perceived data, which is aggregated and processed through Byzantine fault-tolerant protocols executed on-chain. The consensus outcome can then be used to support the enactment of on-chain collective decisions. In this way, Swarm Oracle closes the loop between swarm-level perception and planning, supporting a new blockchain-enabled paradigm for swarm coordination.

Within this paradigm, self-organization emerges not only from individual robot behaviors but also from globally validated information that drives swarm-wide action policies. This information can, for example, regulate resource allocation, update shared databases, trigger behavioral adaptations, and refine learned models. By distributing validation across independent robots, Swarm Oracle ensures reliable operation even when a significant portion of the swarm behaves maliciously. This trustless design makes it suitable for multi-stakeholder deployments, in which

robots operated by different parties can collaborate without needing to trust one another.

The system is evaluated on a collective perception task in which robots estimate environmental colors using biased and noisy sensors. Swarm Oracle achieves correct agreement even when up to one-third of the robots collude to manipulate consensus outcomes. Furthermore, a token-based reputation mechanism enables the system to progressively mitigate the influence of faulty or malicious robots. Through practical application scenarios—including collaborative mapping, decentralized neural network training, and collective foraging—this thesis demonstrates how Swarm Oracle mechanisms can be applied to support coordination and self-organization in realistic scenarios. Although these mechanisms can be deployed on existing blockchain infrastructure, this thesis focuses on the case where the blockchain is maintained directly by the robots themselves. The feasibility of this approach is assessed through experiments with up to 24 physical robots and 120 simulated robots. Results show that storage requirements scale linearly with swarm size and mission duration, while bandwidth, computation, and memory usage remain stable over extended operations.

Overall, this thesis establishes Swarm Oracle as a principled blockchain-enabled mechanism for trustless coordination in robot swarms, providing a foundation for secure, scalable, and self-organizing multi-robot systems in realistic deployments.

Statement of Authorship

This thesis constitutes 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 his supervisor and collaborators, has published in the scientific literature[1]:

General Methods The technical setups and methodologies introduced in Chapter 4 and used throughout the thesis are drawn from Pacheco et al. (2020†, 2022†), where the author was the primary contributor.

Perspective The discussion and figures in Chapter 3 are adapted from Dorigo et al. (2024†), where the author contributed with ideas and text revision. The original work was rewritten, restructured and expanded to situate it within the scope and narrative of this thesis, while remaining faithful to the original perspective.

Proof of Concept The contribution presented in Chapter 5 combines and reinterprets the experiments, results, and discussions from Pacheco et al. (2020†) and Strobel et al. (2023†). In the latter, the author contributed to the conceptualization of the research, setup of the experiments, development and implementation of the robot and simulation software, and revision of the text. The figures from the original contributions are regenerated, and new figures—derived from data collected during the original studies—are added to support the thesis's contributions. The analysis of results and discussion is restructured and expanded accordingly.

Swarm Oracle The research presented in Chapter 6 constitutes the main contribution of this thesis. It corresponds to the paper Pacheco et al. (2025†), where the author was the primary contributor.

Applications The applications presented in Chapter 7 comprise works to which the author contributed either as primary contributor or as co-supervisor of the student who authored the publication. Specifically, Sections 7.2 and 7.3 are based on Pacheco et al. (2024a†, 2022†), where the author was the primary contributor; while Sections 7.4 and 7.5 are based on Moroncelli et al. (2024†) and Van Calck et al.

[1]Throughout the thesis, a dagger symbol (†) in a citation indicates contribution by the author.

(2023†) where the author co-supervised the students, contributing with research guidance and conceptualization, experimental development, results analysis, and manuscript revisions.

Dorigo, M., **Pacheco, A.**, Reina, A., Strobel, V., (2024). "Blockchain technology for mobile multi-robot systems". In: *Nature Reviews Electrical Engineering* 1, pp. 264–274.

Moroncelli, A., **Pacheco, A.**, Strobel, V., Lajoie, P.-Y., Dorigo, M., Reina, A., (2024). "Byzantine Fault Detection in Swarm-SLAM Using Blockchain and Geometric Constraints". In: *Swarm Intelligence – Proceedings of ANTS 2024 – 14th International Conference*. Vol. 14987. Lecture Notes in Computer Science. Springer, pp. 42–56.

Pacheco, A., De Vos, S., Reina, A., Dorigo, M., Strobel, V., (2024a). "Securing Federated Learning in Robot Swarms using Blockchain Technology". In: *Proceedings of the 17th International Symposium on Distributed Autonomous Robotic Systems*. Vol. 34. Springer Proceedings in Advanced Robotics. Springer, pp. 473–488.

Pacheco, A., Strobel, V., Dorigo, M., (2020). "A Blockchain-Controlled Physical Robot Swarm Communicating via an Ad-Hoc Network". In: *Swarm Intelligence – Proceedings of ANTS 2020 – 12th International Conference*. Vol. 12421. Lecture Notes in Computer Science. Springer, pp. 3–15.

Pacheco, A., Strobel, V., Reina, A., Dorigo, M., (2022). "Real-time Coordination of a Foraging Robot Swarm using Blockchain Smart Contracts". In: *Swarm Intelligence – Proceedings of ANTS 2022 – 13th International Conference*. Vol. 13491. Lecture Notes in Computer Science. Springer, pp. 196–208.

Pacheco, A., Zhao, H., Strobel, V., Roukny, T., Dudek, G., Reina, A., Dorigo, M., (2025). *Swarm Oracle: Trustless Blockchain Agreements through Robot Swarms*. arXiv preprint: `2509.15956` (cs.RO). URL: `https://arxiv.org/abs/2509.15956`.

Strobel, V., **Pacheco, A.**, Dorigo, M., (2023). "Robot Swarms Neutralize Harmful Byzantine Robots Using a Blockchain-based Token Economy". In: *Science Robotics* 8.79, eabm4636.

Van Calck, L., **Pacheco, A.**, Strobel, V., Dorigo, M., Reina, A., (2023). "A blockchain-based information market to incentivise cooperation in swarms of self-interested robots". In: *Scientific Reports* 13.20417.

In addition, the author contributed to the following peer-reviewed publications that are not directly integrated into this thesis.

Gupta, H., Strobel, V., **Pacheco, A.**, Ferrante, E., Natalizio, E., Dorigo, M., (2024). "Group-level Behavioral Switch in a Robot Swarm Using Blockchain". In: *Swarm Intelligence – Proceedings of ANTS 2024 – 14th International Conference*. Vol. 14987. Lecture Notes in Computer Science. Springer, pp. 98–111.

Zhao, H., **Pacheco, A.**, Beltrame, G., Liu, X., Dorigo, M., Dudek, G., (2025). "A Blockchain Framework for Equitable and Secure Task Allocation in Robot Swarms". In: *IEEE Robotics and Automation Letters* 10.10, pp. 10862–10869.

Zhao, H., **Pacheco, A.**, Strobel, V., Reina, A., Liu, X., Dudek, G., Dorigo, M., (2023). "A Generic Framework for Byzantine-tolerant Consensus Achievement in Robot Swarms". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS 2023*. IEEE Press, pp. 8839–8846.

The author also developed the following open-source software implementations which have been used in several works by different authors.

Hasselmann, K., Parravicini, A., **Pacheco, A.**, Strobel, V., (2021). *Python wrapper for ARGoS3 simulator*. URL: `https://github.com/KenN7/argos-python/` (visited on 11/11/2025).

Pacheco, A., De Vos, S., Reina, A., Dorigo, M., Strobel, V., (2024b). *Interface `geth-argos`: Trajectory Prediction*. URL: `https://zenodo.org/records/13622916` (visited on 11/11/2025).

Pacheco, A., Denis, U., Zakir, R., Strobel, V., Reina, A., Dorigo, M., (2024c). *Toychain: A Simple Blockchain for Research in Swarm Robotics*. arXiv: `2407.06630 [cs.RO]`. URL: `https://arxiv.org/abs/2407.06630`.

Pacheco, A., Strobel, V., (2020). *Interface `geth-pi-pucks`*. URL: `https://github.com/teksander/geth-pi-pucks` (visited on 11/11/2025).

Pacheco, A., Strobel, V., (2022). *Interface `geth-argos`*. URL: `https://github.com/teksander/geth-argos` (visited on 11/11/2025).

Acknowledgements

This thesis concludes a long and challenging journey, one that would not have been possible without the support of many people. My acknowledgments begin with IRIDIA—a space I have come to appreciate as a second home—and with the people and researchers who make it the best environment I could imagine for completing this journey.

First, my supervisor, close collaborators and friends. Thank you, Marco, for being exactly the supervisor I needed throughout this journey. You pointed me in the right direction every time I needed it; but more importantly, you showed unwavering patience and support whenever I strayed from the supposed path. I admire the way you tackle problems and focus on solutions, irrespective of their original cause.

Volker, I was very fortunate to have you there from the very beginning. It was truly fun to bootstrap and build the foundations for this research together. Not all ideas worked out, but the time spent on the drawing board together was worthwhile. I admire your attention to detail, your patience, and your spot-on feedback and thoughtful criticism each time I came to you for advice.

Giovanni, thank you for your warm support, energy and positivity. There's an argument to be made that my PhD lasted longer than strictly necessary; however, I believe that every additional month or year was worth it. I appreciate the time spent supervising students, Konstanz, and the travels to (un-)conferences—which were totally within research scope. Your mentorship shaped not only this thesis, but many other important aspects of my academic career and life. I admire your open mind and the way you connect with people.

Hanqing, I am really happy our PhD paths crossed. The long days in a dimly lit room with noisy robots were much more enjoyable together. Thank you for hosting me in Montreal and doing everything to ensure I got the full experience. I admire your commitment to your work and the consideration you show to others.

Everyone else in IRIDIA, in some way or another, you have all shaped this thesis and helped make the journey a worthwhile and fulfilling experience. The supervisors for enabling a space that feels like both a lab and a home: Marco, Mauro, Thomas, Hughes, MK. The people who showed me what it means to be

IRIDIAn: Christian, David, Mostafa, Marcolino, Federico, Antoine, Jonas, Weixu, Miquel, Aryo, Yating, Michael, Salman, Guillaume, Cédric. A very special thanks to Ken, Florence, and Alberto, who hosted me in their homes when I needed it.

A different but just as special thank you goes to every IRIDIAn who keeps the flame alive to this day: Guillermo, Lluc, Raina, Giuseppe, Ilyes, Jeanne, Yunshuang, Julien. It is a joy to come into the office, knowing I will find you there for lunch, dinner and drinks. Thanks as well to Alessandro, who came in to check how I am doing just as I write this, Pablo, Paolo, Eduardo and to all those whose time at IRIDIA was meant to be temporary, but whose presence still left a lasting mark.

Also to the students for teaching me more than I taught them: Ludéric, Sebastién, Ulysse, Francesco, Angelo, Dodo, Iacopo and Nemanja.

I am also very grateful to Heiko, Jonas, and Giovanni for the opportunity to be a researcher in Konstanz. Those nine months were an invaluable period in my life and career. Thank you, Heiko, for your immense patience and for your guidance in navigating German bureaucracy. Thanks to everyone in the Cyber-Physical Systems Group for your support and friendship, and to everyone in the CASCB social hours for showing me that it is possible to be a researcher and have a social life.

I also want to thank the people I met doing my two favorite activities at the ULB. The Potager: Thanks to Joel, Yoko and Nicole for pampering me all these years with kindness and food. Thanks to every potagiste for their friendship, and for the many great moments of hard work and fun at the Discosoups. The bras de fer club: Fred, Adrien, Guillaume, Daniel, and others. Thank you for your friendship and for helping me become stronger.

Romain, Nasim, and Jolene: thank you for making Brussels feel like home. I'm so happy I got to go on adventures in Belgium—and beyond—with you. And finally, to my mom, dad, my sister Ana, and my girlfriend Régina: thank you for your support, patience, and love every single day of this journey. A loving thank you to Régina for creating such a special cover for this thesis, and for being by my side through the last, most challenging, stretch of this journey.

— Alexandre —

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The idea of a "hive mind" has long fascinated both naturalists and science-fiction authors. In many novels, films, and games, advanced alien societies are depicted as sharing a collective consciousness under the control of a central leader, such as the Brain Bugs in Starship Troopers that command Arachnid forces, Half-Life's Nihilanth—a creature with a brain so large that it requires external support—, or the Borg Queen who "brings order to chaos" within the Borg Collective in the Star Trek universe. This trope conveniently allows the narrative to hinge on a single point of failure: destroy the leader, and the entire collective collapses. Other science-fiction works, however, explore more distributed forms of intelligence in which every individual is aware of the group's consciousness. In Haldeman's "The Forever War", the thousand-year-long conflict with the Taurans could only end once humanity developed a collective consciousness, in order to communicate with the Taurans' own collective consciousness and realize that the war had always been a long-standing misunderstanding.

In animal groups, however, collective intelligence (Bonabeau et al. 1999, Couzin 2009) arises neither from strict centralization nor from communication among all individuals—which would be infeasible. Ant colonies, for example, have no leader directing the group, and no ant is aware of every other; yet coordinated behaviors such as foraging, nest construction, and defense emerge from simple local interactions and stigmergic communication via pheromones (Grassé 1959). As a result, the colony functions as a superorganism capable of solving complex tasks that exceed the abilities of any single ant. This decentralized organization illustrates a broader principle: by distributing control and information across many agents, the system can continue to function, adapt, and even thrive under adverse conditions, such as environmental stress or partial failure of its components (Albert et al. 2000, Camazine et al. 2020).

These insights have inspired the development of artificial collectives, such as robot swarms, in which desired global behaviors are achieved from the bottom up:

by specifying local control laws at the level of each robot rather than programming a central controller (Beckers et al. 2000, Bonabeau et al. 1999, Trianni et al. 2008). However, designing artificial systems composed of many interacting individuals—while balancing structure and flexibility, robustness and efficiency, and complexity and safety—remains an open challenge. Desired properties are typically specified at the collective level, while the designer intervenes at the level of individual robots, choosing and combining their hardware, control policies, and interaction rules to obtain a collective system that satisfies the original requirements (Brambilla et al. 2013). This process is inherently non-trivial, as the complexity of swarm dynamics can lead to unexpected and potentially dangerous or catastrophic emergent behaviors. Such failures may arise, for instance, from small parametric deviations leading to unpredictable consequences (Hamann 2018), or from a single robot that deviates from its prescribed behavior (Strobel et al. 2018).

Recent work has begun to investigate hierarchical designs (Varadharajan et al. 2024). In particular, Mathews et al. (2017) and Zhu et al. (2024) propose *Mergeable Nervous Systems*, self-organizing hierarchies in which periods of centralization arise to improve coordination or ensure safety during complex tasks. This approach inherently relies on establishing a decision-making hierarchy among the robots.

This thesis pursues an alternative form of hierarchical design, in which each robot contributes equally to a collective memory and decision-making process. This is accomplished through the integration of blockchain technology with swarm robotics, a recent proposal to address security challenges in swarm robotic systems (Castelló Ferrer 2018, Strobel et al. 2018).

Blockchain technology, first brought to prominence by Nakamoto (2008), has since become an active research area in distributed systems and cryptography, investigating mechanisms for achieving global consensus in decentralized networks without central leadership. A blockchain network synchronizes a shared memory state that can be trusted even when participants do not trust one another and communicate on a best-effort basis. This memory can, among other things, be used to record interactions among agents, store data, and enact collective decisions through smart contracts (Szabo 1997)—self-executing programs whose operation cannot be censored, halted, or manipulated by any single participant. In swarm robotics, Strobel et al. (2020) used smart contracts to enhance robustness against adversarial or faulty robots—known as Byzantine robots—that behave erratically or act against the objectives of the collective (Bouzid et al. 2010, Lamport et al. 1982). Building on this idea, the present work advances the state of the art in the security and design of robot swarms. This thesis applies blockchain technology not only to withstand and mitigate the negative impact of Byzantine robots, but also to enable new design methodologies for achieving self-organization and coordination in swarm robotics.

Yet this ambition encounters a fundamental limitation of blockchain systems: the *oracle problem* (Antonopoulos and Wood 2018). While smart contracts can enact collective decisions over data already recorded on-chain, they cannot access external information from the physical world. However, swarm decisions often depend on robots' perception of the environment—for example, the location of resources or whether a task has been completed. Any device, algorithm, or mechanism that supplies such external information to a blockchain is called an *oracle*; in a robot swarm, however, delegating that role to a single robot or external service would reintroduce precisely the trusted intermediary and potential single point of failure that decentralization seeks to avoid.

The central proposition of this thesis is therefore the concept of *Swarm Oracle*: a system in which the swarm itself collectively fulfills the role of oracle by transforming the noisy local observations of many robots into global facts that smart contracts can use for decision-making. In this way, swarm *self-organization* no longer arises solely from local robot-to-robot interactions, but also from the feedback loop between distributed perception, on-chain validation, and collective rules encoded in smart contracts. The result is a *trustless* model of swarm self-organization, whereby collective decisions are not grounded in the assumed reliability of any single robot, human operator, or external authority, but instead emerge from distributed validation of facts and shared rules enacted through blockchain-based protocols.

1.1 Research contributions and thesis structure

The development of this thesis coincided with several other research and industry efforts to integrate blockchain technologies across different sectors. Initiated in October 2019, this thesis identifies and narrows the literature and methodological gaps between swarm robotics and blockchain technologies, paving the way for the integration and advancement of the two fields. The following paragraphs summarize the structure and the scientific contributions presented in this thesis.

A background study bridging swarm robotics and blockchain. Chapter 2 synthesizes the core principles that underpin the two fields as well as the fundamental differences that distinguish them, generating a common vocabulary. In particular, it discusses *consensus achievement* as the fundamental building block of self-organization, approached differently across the two fields. The background is written to orient readers expert in one field but new to the other.

A critical review of the state of the art in key challenges. Chapter 2 additionally reviews the state of research in key challenges. In particular, it reviews swarm robotics' literature on fault-detection, robustness, and resilience to the

presence of Byzantine robots; and in blockchain, it reviews existing developments that address the scalability and oracle problems. Scalability refers to the trade-offs involved in maintaining the desired latency and data throughput as the network grows, without imposing prohibitive storage, bandwidth, or computational costs on nodes. The oracle problem concerns the challenge of introducing externally observed facts into a deterministic blockchain without reintroducing trusted intermediaries to supply information that other nodes cannot independently verify. Finally, existing deployments of blockchain-based robotics systems are reviewed, in particular, prior work focusing specifically on swarm robotics.

A perspective that frames central research questions. Chapter 3 presents a perspective on the integration of blockchain and swarm robotics, developed over nine years of research at IRIDIA, which encompasses six and a half years of the authors' doctoral studies. It builds on the challenges and opportunities introduced in Dorigo et al. (2024†), reorganizing them to frame the central research questions addressed in the following chapters and motivate the work developed in this thesis.

A proof-of-concept study that demonstrates feasibility and scalability. Chapter 5 addresses the hardware feasibility and scalability concerns of blockchain integration in swarm robotics. It presents results from the first blockchain deployment in a physical robot swarm (Pacheco et al. 2020†), later extended to larger scales (Strobel et al. 2023†), with swarms composed of 24 physical and 120 simulated robots. While Strobel et al. (2023†) primarily focuses on securing consensus achievement in robot swarms, this chapter shifts the emphasis toward the feasibility and scalability of deploying blockchain technology in swarm robotic systems, which constituted the main contributions of the thesis author to this body of work. In particular, Chapter 5 extends the original studies with new analyses of blockchain scalability and hardware costs, which were not discussed in the original publication but are derived from data collected during the experiments. These include physical measurements of the storage, bandwidth, computation, and runtime memory requirements on robots across swarm sizes and experiment duration. This contribution can be viewed as a proof-of-concept robotic oracle network, in which the robots serve as oracle nodes that report sensor-derived estimates of the frequency of white floor tiles to a smart contract. As a proof of concept, the approach is limited to a specific scenario and a one-dimensional estimate, without formal guarantees preventing local disturbances from affecting swarm-level consensus.

Addressing the oracle problem of blockchains. Chapter 6 presents the main contribution: Swarm Oracle, a framework that addresses the blockchain oracle problem through a swarm of robots that collect, exchange, and reach consensus on data perceived from the physical world. In contrast with swarm robotics'

local and cooperative consensus protocols, Swarm Oracle adopts a trustless design based on Byzantine fault-tolerant protocols and cryptography. In this approach, each robot independently validates information and performs the computations required for consensus, mitigating the risk that errors propagate through peer-to-peer interactions. This provides the basis for theoretical guarantees of safety and liveness (Lamport 1977): that consensus is correct and that it is eventually achieved, as long as the number of Byzantine robots remains below the established limits. Extensive experiments assess how these guarantees translate under realistic sensing, motion, and communication constraints; and how Swarm Oracle withstands coordinated attacks by up to one-third of the swarm, while a reputation-based resilience mechanism mitigates the effects of attacks and faults over time. By supplying real-world data to a blockchain through decentralized consensus among robots, Swarm Oracle addresses the oracle problem assuming requiring trust in any single robot. In other words, robots may deviate from established protocols or act in self-interest. This is valuable both for swarm self-organization, where blockchain-based collective decisions should not depend on a single robot's data, and for multi-stakeholder swarms or public blockchain applications, where trust should not be placed in any single robot stakeholder, manufacturer or operator.

Implementation and critical analysis of applications. Chapter 7 demonstrates that blockchain technology can, not only secure consensus achievement in swarm robotics, but also expand the capabilities of decentralized swarms of simple robots in practical applications. In each application, a shared database, synchronized across the swarm using blockchain protocols, is continuously updated with information provided by oracle mechanisms tailored to the application. This is demonstrated across four applications: task allocation and foraging (Section 7.2), federated learning (Section 7.3), collective mapping (Section 7.4), and social navigation (Section 7.5). Each application discusses the underlying oracle problem, the oracle mechanisms that secure on-chain information, and how this information is leveraged to organize swarm behavior. The results assess core swarm properties: fault tolerance, scalability, and flexibility. Through simulation studies, this contribution validates a novel self-organization paradigm for swarm robotics, in which swarm behaviors are steered by on-chain facts constructed through decentralized consensus and oracle mechanisms, advancing the state of the art in the capabilities of robot swarms and narrowing the gap toward real-world applications.

Chapter 2

Background

This chapter introduces the two main research fields underpinning the thesis: *swarm robotics* and *blockchain technology*. Both fields study decentralized systems composed of autonomous agents—software nodes or robots—that rely on consensus and self-organization to achieve desirable system-wide properties. By design, these systems aim to minimize costs and dependence on any particular agent, using peer-to-peer communication to avoid single points of failure and mitigate communication and computation bottlenecks. Self-organization, whereby global order emerges from local interactions rather than centralized control, underpins scalable collective behavior. Accordingly, both fields adopt bottom-up design, specifying behaviors, incentives, and capabilities at the individual level to preserve collective properties as the system size increases.

These parallels motivate a central research question of this thesis:

How can knowledge and technologies be transferred between blockchain systems and swarm robotics?

Despite these parallels, the two fields differ in several fundamental aspects. Most notably, robots interact not only through digital communication but also via the physical environment, creating a mismatch between blockchain's deterministic execution and the inherent stochasticity of physical interactions. Another key distinction lies in behavioral assumptions: swarm robotics generally assumes cooperative agents—with deviations arising from noise, faults, or by-design behavioral heterogeneity—whereas blockchains operate in open networks where peers may act strategically due to self-interest or malicious intent.

These differences lead to different approaches to *consensus achievement*—a fundamental primitive of coordination and self-organization in distributed systems. In swarm robotics, consensus is typically social, whereby individual robots' states are influenced by neighbors until the swarm converges to an approximate collective

value or decision. By contrast, blockchain consensus aims to establish a global history of state transitions that is replicated by each agent. While temporary disagreements are acceptable, a blockchain network should eventually converge to an exact state history that is agreed upon by every agent. In both fields, consensus arises among gossiping peers and relies on probabilistic convergence towards a common state rather than deterministic guarantees. This concession is crucial for ensuring that communication costs scale efficiently with the number of agents.

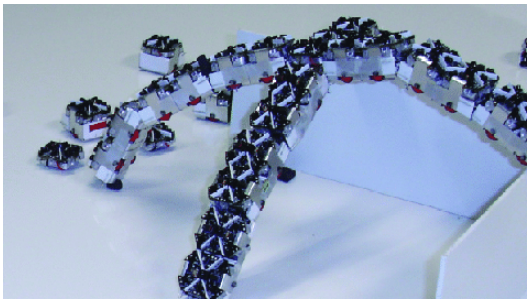
Another important distinction concerns *behavioral alignment* and *trust*. In swarms, individuals execute controllers and behaviors to realize a common objective, whereas in blockchain systems coordination emerges from incentive structures and game-theoretic mechanisms that align the behaviors of untrusting participants.

The remainder of the chapter is structured as follows. Sections 2.1 and 2.2 introduce the foundations of swarm robotics and blockchain technology, respectively. For each field, they introduce (i) system organization and design principles, (ii) consensus achievement, and provide a literature review on (iii) state-of-the-art approaches and challenges relevant to this thesis. Concretely, for swarm robotics, this review covers bio-inspired robustness, fault detection and mitigation, and protections against Byzantine robots. For blockchain systems, it surveys scalability and the oracle problem—the challenge of incorporating noisy, uncertain, or localized real-world information to a deterministic blockchain without compromising decentralization. Finally, Section 2.3 connects both fields by presenting system architectures for integrating robotics and blockchain systems, and reviewing the state of the art for each architecture.

2.1 Swarm robotics

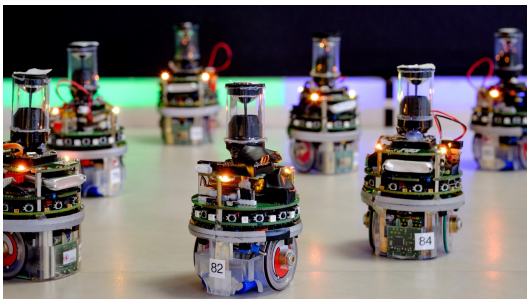
Swarm robotics studies how principles of swarm intelligence can be embodied through robotic systems to achieve desired collective behaviors. It establishes a design paradigm for robotic systems inspired by the observation and understanding of natural collectives, such as eusocial insects or the coordinated motion of schools, herds, and flocks (Dorigo et al. 2014). Instead of relying on a centralized controller or external infrastructure, a swarm leverages the autonomous actions of many relatively simple robots to collectively accomplish tasks beyond the capabilities of any individual member. Through studying how simple local interactions yield complex collective behaviors, swarm robotics has deepened our understanding of collective intelligence in both natural and artificial systems. While practical deployments remain limited, future applications are expected in scenarios demanding high autonomy and scalability, for example, in agriculture, disaster response and underwater or extraplanetary exploration (Dorigo et al. 2020). Figure 2.1 shows the robotic platforms used over recent decades of swarm robotics research.

(a) Alice (Caprari et al. 1998)



(b) Jasmine (Kornienko et al. 2005)

(c) Swarm-bots (Mondada et al. 2004)



(d) E-pucks (Mondada et al. 2009)

(e) Kilobots (Rubenstein et al. 2012)

(f) Pogobots (Loi et al. 2025)

(g) TurtleBot4 (Clearpath Robotics 2025)

Figure 2.1: Robot platforms used in swarm robotics research. Image credits: (a) Wikimedia Commons, (b) IPDS, Stuttgart, (c–e) IRIDIA, Brussels, (f) ISIR, Paris (Photo: N. Bredeche), (g) CASCB, Konstanz (Photo: E. Böker).

2.1.1 System organization and design

Robot swarms are self-organizing multi-robot systems in which each robot acts in response to its perception of the environment and its interactions with neighboring robots. Decentralization and peer-to-peer interactions are key to achieving scalable, flexible and robust collective behaviors: by avoiding a central element that could act as a communication bottleneck or a single point of failure, swarms can continue operating even when many robots are lost and communication links fail.

Designing a robot swarm typically follows a *bottom-up* approach, in which developers iteratively formulate individual controllers and interaction rules to generate desired global behaviors. Owing to the complexity of this approach, much early research in swarm robotics focused on specific collective behaviors inspired by organizational patterns observed in nature. Trianni and Campo (2015) provide a detailed overview of these behaviors, including bee-inspired decision-making (Reina et al. 2015) and ant-inspired foraging (Campo et al. 2010). In such work, designers often exploit behavioral primitives that correspond to recurring interaction patterns (for example, aggregation, dispersion, consensus, or task allocation) and then combine them into higher-level controllers. However, engineering artificial swarms that are capable of solving complex real-world tasks remains a challenging process that relies on the designer's intuition and trial-and-error experimentation. As tasks and environments become more complex, this process becomes time-consuming and difficult to reproduce.

To address these limitations, *automatic design methods* leverage simulation and optimization to synthesize individual behaviors without iterative manual refinement. A prominent class of approaches uses evolutionary algorithms that search the space of controllers—often represented as neural networks or other parametric structures—to optimize task performance in simulation (Nolfi and Floreano 2000). Controllers generated in this manner can then be transferred to real robots for validation (Dorigo et al. 2004). AutoMoDe (Francesca et al. 2014) is another approach to automatic design that uses libraries of well-studied behavioral modules, which are combined and parametrized to synthesize robot control software that maximizes a given objective function.

Although a significant portion of swarm robotics research focuses on horizontally organized collectives, recent studies show that *heterogeneity and explicit system structures* can enhance swarm capabilities without degrading its desired properties. The Swarmanoid project, for example, demonstrates that a heterogeneous swarm composed of foot-bots, hand-bots and eye-bots can perform complex tasks by exploiting the robots' complementary capabilities without centralized control (Dorigo et al. 2013). Mathews et al. (2017) introduced the concept of a mergeable nervous system, a control paradigm in which robots can dynamically form physical connections to form larger composite bodies controlled by a single "brain" robot.

Zhu et al. (2024) extend this concept, enabling physically independent robots to build self-organized control hierarchies in which localized leaders coordinate sensing, actuation, and decision-making, while maintaining scalability and fault-tolerance.

Properties By following these organizational and design principles, robot swarms are inherently endowed with certain desirable *properties*. Nevertheless, guaranteeing or enhancing these properties in real applications often requires researching and developing dedicated mechanisms (Hamann 2018). Dorigo et al. (2014) identifies *scalability*, *flexibility*, and *fault tolerance* as defining properties of a robot swarm. In this thesis, *robustness* and *resilience* are also relevant because they provide a more fine-grained characterization of *how* robotic systems respond to faults (Prorok et al. 2021). The definitions adopted in this thesis are shown in Box 2.1.

Box 2.1: Properties of swarm robotic systems

Scalability Changing the number of robots allows the system to tackle proportionally sized problems while maintaining performance.

Flexibility The swarm's ability to solve modified problems or maintain performance in environments different to those considered during design using the same underlying control mechanisms.

Fault tolerance The swarm's capability to remain operational in the presence of faults that occur in robots or other system components.

The concepts of robustness and resilience further characterize the systems' response to the presence of faults:

Robustness The swarm withstands faults without needing to adapt its structure or behavior by exploiting built-in redundancy, protection mechanisms, or safety margins, to exhibit a graceful degradation in performance.

Resilience The swarm is able to recover from faults by adapting its strategies and behaviors or by reconfiguring itself, restoring performance levels even after a severe initial degradation in performance.

In the robotics literature, robustness can—in addition to internal component faults—be associated with environment disturbances or noise stemming from external sources (Dorigo et al. 2021, Şahin 2005); the spreading of incorrect social information (Antonic et al. 2024); or the handling of mismatches between assumed and actual system models—including variations in network topology, model parameters, or robot behaviors (Gupta et al. 2006, Nguyen and Sreenath 2022, Zhang et al. 2015). Studies of resilience go even further, considering extreme events such as adversarial attacks and long-term network partitioning (Gil et al. 2017, Saulnier et al. 2017).

In this thesis, the usage of the terms robustness and resilience deliberately abstracts away from the source or cause of the faults, and instead focuses on how the swarm copes with their presence and negative effects, as portrayed in Figure 2.2. In part, this stems from the fact that this thesis studies *Byzantine faults* (see Section 2.1.3 – C), which are erratic and cannot be reliably linked to any particular source: they may arise from hardware or software faults, but also from noise, misleading social information, localized environmental disturbances, or adversarial tampering. As concrete examples, the term *Byzantine fault tolerance* is used to denote a swarm's capability to withstand the presence of Byzantine robots; blockchain-based protections are presented that enhance the robustness of swarms, with results showing how changes in the number or behavior of Byzantine robots affect performance; and reputation mechanisms and economic incentives enhance swarm *resilience* by adaptively regulating and mitigating the influence of Byzantine robots on collective outcomes.

The work in this thesis shares a similar background with LeBlanc et al. (2013) and Zhang and Sundaram (2012b), who study consensus in the presence of faults and adversarial agents in a distributed network (see Section 2.1.2). However, their use of the terms robustness and resilience differs from the usage adopted here. In their work, *robustness* is defined as a graph-theoretic property of the communication network (r-robustness) that quantifies how strongly a given subset of nodes is connected to the rest of the network.[1] LeBlanc et al. (2013) leverage assumptions on network robustness to derive bounds on the number of adversarial or faulty agents that can be present while guaranteeing that their *resilient* consensus algorithm (W-MSR) will converge toward correct values. These notions of robustness and resilience are not directly applicable to swarm robotics, as they assume particular network structures and spatial distributions of faulty agents that are difficult to guarantee in large-scale outdoor deployments. Moreover, the resilient algorithms proposed by the authors in this line of work rely on fixed update rules and do not incorporate adaptation or reconfiguration mechanisms.

[1] For example, in a network where a single robot is the only bridge between two sub-networks, the network can be at most 1-robust.

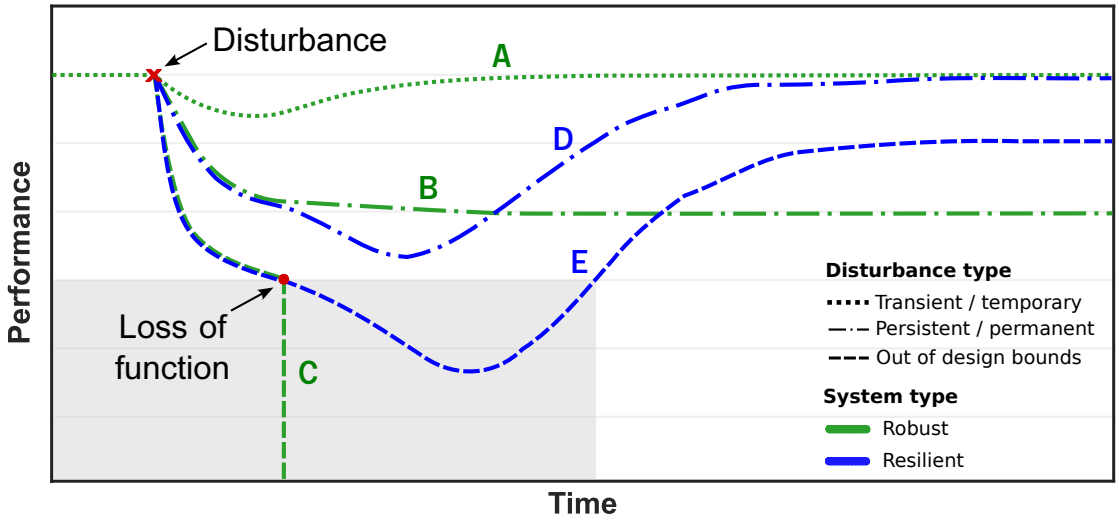


Figure 2.2: Robustness versus resilience. **(A)** *Robust systems* (green lines) tolerate temporary disturbances without structural change and without lasting loss of performance. **(B)** Under persistent or permanent disturbances, robust systems' performance may degrade since the system structure remains unchanged. **(C)** If disturbances exceed design limits, robust systems suffer loss of function (gray area). **(D)** *Resilient systems* (blue lines) adapt to persistent disturbances to gradually recover operational performance over time. **(E)** Through self-reconfiguration, resilient systems may recover from an initial loss of function and settle into a new operational regime. The shown curves are schematic and do not represent any particular system.

2.1.2 Consensus achievement

Reaching collective agreements in a distributed system is often the first step towards coordination or self-organization. Robotic collectives first have to reach agreements on the state of the environment or of their operations so that they each may compute a common plan and act cohesively.

As such, consensus achievement has been extensively studied in mobile multi-robot systems, especially in swarm robotics. Most approaches explicitly avoid centralized servers or global communication among robots which would otherwise undermine fault tolerance and scalability. Instead, they rely on local interactions among robots which individually process information received from its neighbors to update their individual state. This naturally leads to asynchronous system models, where communication delays, intermittent connectivity, and non-simultaneous updates are treated as intrinsic features of the system.

From classical distributed computing, it is known that achieving system-wide

deterministic consensus in asynchronous systems with unreliable communication and possible faults is a major challenge, and in some settings provably impossible (Fischer et al. 1985). For this reason, consensus algorithms for multi-robot systems typically focus on *approximate* consensus, in which agents iteratively adjust their states and gradually converge toward a common agreement, which terminates when robot opinions are within an ϵ-threshold of one another.

The remainder of this section introduces well-studied consensus problems in robotics, including the discrete **best-of-n decision making** problem in swarm robotics; and continuous problems such as **multi-robot rendezvous**, where robots observe each others positions to coordinate their motion, and **collective sensing** problems, where robots exchange and agree on values they individually observe in the environment.

(A) Best-of-n decision making

Decision making in swarm robotics is inspired by natural collectives, observing, for example, how honeybees select nest sites (Reina et al. 2014a) or how ants choose between different paths (Campo et al. 2010).

Valentini et al. (2017) formalize this type of consensus as a best-of-n problem, in which the opinions exchanged among robots are discrete choices in a finite set of options. Common bio-inspired strategies include the *voter model*, in which robots adopt the opinion of a randomly selected neighbor, the *majority rule*, where robots adopt the option supported by the local majority, or *direct modulation*, in which robots promote options in proportion to their perceived quality. Valentini et al. (2015a) show that bio-inspired strategies exhibit a trade-off between convergence speed and decision accuracy. For example, compared with the voter model, the majority rule converges more quickly but is more vulnerable to early fluctuations and thus more likely to settle on a suboptimal choice.

(B) Multi-robot rendezvous

Robots may also need to reach agreements on continuous values, such as directions or velocities for group motion (Ferrante et al. 2012). In rendezvous problems (Roy and Dudek 2001), robots agree on a common position in space *without exchanging explicit messages*. Instead, robots directly observe the positions and velocities of their neighbors, which serve as inputs to their local motion control laws.

This problem has been studied both in idealized theoretical settings, where agents have global knowledge of the positions and velocities of all other robots (Jadbabaie et al. 2003, Vicsek et al. 1995), and under more realistic assumptions of localized perception, in which robots can only observe nearby neighbors (Cortés et al. 2006, Flocchini et al. 2005, Lin et al. 2003). Another common modeling

assumption concerns the robots' capabilities. Some models assume *oblivious* robots, which do not retain memory of previously perceived information (Flocchini and Prencipe 2012), while others allow robots to maintain an internal state that persists over time (Cieliebak 2004). Frequently, models also assume that robots are anonymous, lacking the ability to uniquely identify or distinguish one another (Flocchini et al. 2008).

The execution model governing robots' control cycles is another key modeling assumption in rendezvous problems. In general, robots follow a Look–Compute–Move control cycle, in which they observe the positions and/or velocities of neighbors, compute a control action, and then move accordingly. Different execution models specify how these cycles are scheduled across robots. In the ATOM (semi-synchronous) model, robots complete each Look–Compute–Move cycle in a single atomic step, such that they can only be observed at their final positions (Suzuki and Yamashita 1999). In contrast, in the asynchronous CORDA model, robots may observe others while they are mid-cycle, as each robot executes its controller independently and at its own pace (Prencipe 2005).

(C) Collective sensing

Collective sensing enables a team of robots to form a shared estimate of an environmental quantity from the many noisy measurements sampled by individual robots. To do so, robots *exchange locally computed values with neighbors* and iteratively reconcile them in a message-passing system (Morlino et al. 2010). Each robot's values are unique and internal to the robot itself, since they reflect the robot's local processing of its own sensor measurements and values received from others. Several advances in robotics collective sensing stem from research in multi-agent systems, which model similar message-passing of noisy values among agents in networks (LeBlanc et al. 2013, Olfati-Saber et al. 2007, Zhang and Sundaram 2012a).

Olfati-Saber and Murray (2004) presented the linear consensus protocol (LCP), in which agents average the values received from graph neighbors, while Zhang and Sundaram (2012a) proposed the median consensus algorithm (MCA) in which each agent uses a weighted combination of its own value and the median of its neighbors' values, yielding estimates which are more tolerant to noise. LeBlanc et al. (2013) introduce the weighted-mean-subsequence-reduced (W-MSR) consensus algorithm, which further improves tolerance to noise and biases induced by faulty agents. In W-MSR, each robot orders the values received from peers (including its own), and then discards the f smallest and the f largest values, where f is a design parameter that imposes limits on the number of faulty agents. The authors then derive theoretical conditions under which normal agents are guaranteed to achieve consensus. These conditions are expressed in terms of network robustness (Zhang

et al. 2015), meaning that if the network is not sufficiently robust (i.e., when there are insufficient redundant communication links among sub-networks), the presence of faulty agents may impede consensus. In a large-scale robot swarm, where communication is intermittent and localized, network structures are not guaranteed at all times during operation. Yu et al. (2021) considers event-based communication triggered by physical meetings among robots. Their work leverages robots' periodic motion to obtain the needed network robustness, and achieve consensus by augmenting W-MSR with a sliding-window in which robots keep a memory of recent messages before trimming extreme values. This approach can enhance convergence guarantees, particularly when robots display periodic motion patterns in a shared environment.

2.1.3 State of the art

The previous sections described how swarm robotics' are designed and organized to achieve desired properties such as fault-tolerance, robustness, resilience and scalability. Decentralization, local communication, and bottom-up design are widely regarded as prerequisites for these properties (Dorigo et al. 2014, Garattoni and Birattari 2016, Hamann 2018). However, Bjerknes and Winfield (2013) challenges the common assumption that swarm robotic systems are robust and scalable by design. In their experiments, they show that worst-case partial faults or disturbances can cause system reliability to drop rapidly as swarm size increases, motivating new approaches to achieve desired properties at scale.

Because swarm behaviors are deeply shaped by information exchange and repeated interactions among many robots, local faults can propagate through the collective and give rise to unsafe emergent behaviors posing potential risks to humans, animals, other robots, and the environment (Gielis et al. 2022, Hunt and Hauert 2020). In adversarial settings, system vulnerabilities or weaknesses can become acute points of failure: Strobel et al. (2018) showed that even a single malicious robot attacking the network can halt the operation of an entire swarm.

As swarm size and autonomy increase, continuous external monitoring becomes infeasible, so fault tolerance, robustness and resilience must be engineered as system-level properties to enable safe real-world deployments (Hunt and Hauert 2020). However, most approaches in swarm robotics assumes cooperative robots or that faults can be accurately modeled at design time. In real deployments, swarms may face unmodeled disturbances, partial failures, or adversarial attacks that violate these assumptions and are difficult to detect reliably.

The remainder of this section surveys state-of-the-art methods for tolerating faults and disturbances in swarm robotics, organized around three themes: **bio-inspired robustness**, **fault detection and mitigation**, and **protection against Byzantine robots**.

(A) Bio-inspired robustness

Bio-inspired swarm robotics seeks to achieve high levels of robustness using nature's guiding principles, such as redundancy and simple local policies among agents with limited capabilities (Dorigo et al. 2014, Garattoni and Birattari 2016, Hamann 2018). Classic examples include bee-inspired collective decision-making (Reina et al. 2014a), ant-inspired exploration using artificial pheromones (Ducatelle et al. 2010), and cockroach-inspired aggregation (Garnier et al. 2009).

A substantial body of work has investigated how observed natural behaviors can be algorithmically replicated to enhance robustness in consensus achievement and collective decision making when robots exhibit non-ideal behaviors. Studied deviations include zealot or stubborn robots that refuse to change their opinion (Antonic et al. 2024, Masi et al. 2021, Prasetyo et al. 2018) and contrarians that systematically oppose social cues (Khaluf and Hamann 2019). Cross-inhibition—a voting model in which agents cancel others' signals, derived from observations of foraging honeybees (Pais et al. 2013, Reina et al. 2015)—has been shown to improve robustness in robot swarms. Specifically, Reina et al. (2023) show that it can help overcome individual biases, Talamali et al. (2019) observed improved decision accuracy, and Zakir et al. (2024a) demonstrate that it can break decision deadlocks.

Researchers also study how non-idealities (e.g., noise, biases or limited perception) that are naturally present in both artificial and natural swarms can, under certain circumstances, help to improve robustness. Liu and Passino (2004) investigated how stable foraging behaviors emerge despite high levels of noise in individual perception. Talamali et al. (2021) show that reduced communications can limit the spread of erroneous information, improving the swarms' ability to adapt to environmental changes; Zakir et al. (2024b) show that errors in information transfer can increase the accuracy of swarm decisions; and Raoufi et al. (2023) show how deviations among agents are not necessarily a bug or a flaw, but rather a feature that can be exploited to improve performance. They present counter-intuitive results where, in a sufficiently large swarm, robots with uncalibrated motion actuators can perform better than calibrated ones.

However, as noted by Winfield and Nembrini (2006), individuality, decentralization, and redundancy do not by themselves guarantee fault tolerance in realistic deployments. Bio-inspired algorithms can be vulnerable to problems such as digital double counting (where outdated or duplicated information is repeatedly treated as new) or severe noise that propagates through stigmergic variables or social feedback. These problems may affect the swarm's ability to coordinate, for example, by disrupting stigmergic trails (Aswale et al. 2022) or leading to biased decisions (Masi et al. 2021). These limitations motivate dedicated research toward developing mechanisms to detect faults in robots and to mitigate their adverse effects.

(B) Fault detection and mitigation

Fault detection aims to identify and isolate malfunctioning robots before their behavior degrades collective performance or compromises safety. Robot swarms provide a natural foundation for fault detection because many robots can cross-validate information and peer behaviors more effectively. Fault detection strategies in swarm robotics are commonly categorized as *endogenous* or *exogenous*, depending on whether faults are identified by the affected robot itself or by its peers (Christensen et al. 2008).

Endogenous methods rely on self-monitoring, where robots detect discrepancies between expected and observed signals using either explicit models (Isermann 2005, Roumeliotis et al. 1998) or data-driven representations of their nominal operations learned from experience (Christensen et al. 2008, Tarapore et al. 2017). Such techniques are well suited to *partial faults*, in which a robot's performance drops due to sensor or actuator degradation. However, they are ineffective under *crash faults*, where robots shut down and therefore cannot self-monitor, and they can also fail when faults fall outside the scope of available models or learned data representations.

In exogenous fault detection, robots observe each other to identify inconsistencies indicative of partial or crash faults in actuation, sensing, or communication. Inspired by synchronized flashing in fireflies, Christensen et al. (2009) introduced a decentralized scheme in which robots propagate liveness/activity signals to detect crashed peers. To detect partial faults, Lau et al. (2011) exploit task-level performance comparisons, for example by monitoring discrepancies in foraging efficiency among robots performing identical roles. More recently, Lee et al. (2022) extract behavioral metrics from swarm-level data to identify anomalous individuals. Immune system-inspired approaches (Tarapore et al. 2017, 2015, 2019) propose online learning mechanisms to discriminate normal from abnormal peer behavior while maintaining tolerance to legitimate shifts in nominal swarm behavior.

Once a fault or abnormal behavior is detected, mitigation mechanisms aim to contain its impact and preserve collective function. Mitigation can include isolation (e.g., ignoring messages or cues from abnormal robots), physical isolation or removal, or network or hierarchical reconfiguration. For example, Christensen et al. (2009) propose physical removal of crashed robots by operational peers, reducing physical obstructions that interfere with swarm operations. Timmis et al. (2016) mitigate the impact of active faulty robots by removing them from distributed algorithms to prevent further degradation. Related work proposes constructing "artificial granulomas" to physically contain faulty robots (Khadidos et al. 2015). In another line of work, Oğuz et al. (2025) propose a proactive–reactive strategy in which the swarm agrees in advance on a recovery plan to restore connectivity; this plan can be enacted when intermittent faults are detected and retracted once normal

operation resumes.

However, most decentralized detection and mitigation schemes implicitly assume that robots follow predefined protocols and honestly report observations or diagnostics. Under these assumptions, they are effective for deviations that are observable to peers, but their effectiveness degrades in realistic deployments where faults may be intermittent or unmodeled, leading to unexpected and erratic robot behaviors that break models and data-driven algorithms. This limitation becomes particularly salient under adversarial conditions, in which compromised robots may maliciously mimic nominal behavior while exploiting vulnerabilities, or report incorrect information to induce erroneous detection. This motivates the need for protection mechanisms to safeguard robot swarms against Byzantine faults, which cannot be reliably detected. This motivates yet another central research question for this thesis, presented in Box 2.2.

Box 2.2: Byzantine faults and trustless swarm design

The potential presence of Byzantine robots which cannot in general be reliably identified by other robots in a swarm contrasts with the traditional cooperative nature of swarm robotics, highlighting a central research question of this thesis:

How can a robot swarm be designed when robots do not trust one another?

While this question arises naturally from the need to engineer swarms that can tolerate Byzantine robots, this thesis goes one step further: robots are explicitly allowed to exhibit strategic or self-interested behaviors that may not be aligned with the collective objective. In Chapter 3 the concept of *open swarms* is discussed, composed of robots that belong to different stakeholders pursuing different individual goals. In Chapter 6 a swarm application developed under a trustless system design paradigm is presented. In Chapter 7 – Section 7.5 an information marketplace is presented, where cooperation among robots is not assumed but instead emerges from robots that maximize individual profits.

(C) Protection against Byzantine robots

The concept of Byzantine faults originates from the Byzantine Generals problem in classical distributed computing (Lamport et al. 1982), which highlights the difficulty of achieving consensus in distributed systems when some participants display erratic, malicious or strategic behaviors.

In the robotics literature, the term *Byzantine robot* refers to a faulty robot that can deviate arbitrarily from its prescribed algorithm. Such a robot may, for example, move erratically, disseminate false or inconsistent messages, omit messages, or exploit system vulnerabilities. Early robotics work already recognized

the need to study coordination under Byzantine behavior. In particular, Peleg (2005) characterizes Byzantine robot behavior as "arbitrary and unforeseeable," as if robots were under the control of a malicious adversary seeking to degrade the system. In real applications, Byzantine behaviors may arise from deliberate attacks, but also from unmodeled faults, unanticipated environmental interactions, or complex failure modes that violate the assumptions and algorithms used by conventional detection and robustness mechanisms. The potential presence of Byzantine robots, which cannot in general be reliably identified by their peers in a swarm, raises the research question in Box 2.2. Consequently, protecting a swarm against Byzantine robots typically requires dedicated design or mechanisms that limit or completely neutralize the impact of Byzantine robots.

The following paragraphs discuss robustness to Byzantine agents in multi-robot rendezvous, message-passing systems, and blockchain-enabled robotic systems.

Multi-robot rendezvous The impact of Byzantine robots has been thoroughly studied in the multi-robot rendezvous (or gathering) literature. In this line of research, Byzantine robots typically change their behavior or motion pattern in ways that disrupt the computations of other robots observing their position. Agmon and Peleg (2006) present a deterministic algorithm that solves the Byzantine gathering problem in a system of N robots including f Byzantine robots as long as $N \geq 3f + 1$. Their solution, however, remains impractical for swarm applications due to its complexity, synchronous requirements, and all-to-all communications. Asynchronous models such as CORDA are more representative of swarm deployments, since each robot observes its neighbors, computes, and moves at its own pace, without synchronizing with others (Prencipe 2005). Using this model, Agmon and Peleg (2006) show that deterministic gathering becomes impossible in the presence of even a single Byzantine robot. This motivates the search for weaker solutions to the problem, typically termed *probabilistic gathering* in which robots asymptotically converge towards common coordinates. Following this direction, Défago et al. (2006) established feasibility of probabilistic gathering under varying assumptions about synchrony and fault types, and a sequence of works by Bouzid et al. (2009a,b, 2010) proposed Byzantine-tolerant gathering algorithms for asynchronous systems under different assumptions on robot capabilities and neighbor observability. Collectively, these works tackle the fundamental challenges of Byzantine tolerance in robotics, while also highlighting the gap between theoretical feasibility and the constraints of large-scale swarms.

Message-passing systems Gathering problems do not fully capture the potential harm of Byzantine agents in message-passing systems. When robots coordinate through explicit communication, Byzantine robots can transmit arbitrary and mutually inconsistent messages to different neighbors. In consensus protocols over communication graphs, LeBlanc and Koutsoukos (2011) distinguish *malicious*

robots that broadcast the same untruthful value to all neighbors at a given time step from *Byzantine* robots that may additionally send different values to different neighbors simultaneously.

A large body of work seeks to provide provable consensus guarantees despite the presence of adversaries in distributed networks. Examples include overcoming malicious links in wireless dissemination (Sundaram et al. 2012), compensation strategies under transient and intermittent faults (Parlangeli 2013), and a complete analysis of the challenges of performing distributed computations in the presence of malicious agents (Sundaram and Hadjicostis 2008a,b). A recurring theme across these approaches is that correctness depends not only on the update rule but also on the redundancy and robustness of the communication network in order to ensure that Byzantine agents are unable to target isolated sub-networks and manipulate consensus achievement. LeBlanc et al. (2012) contrast *f-local* models, which bound the number of Byzantine agents in each agent's neighborhood, with *f-global* models, which bound the total number of Byzantine agents in the system. While such models may yield strong guarantees under sufficiently r-robust networks (Zhang et al. 2015), they also expose a key challenge for swarm robotics: maintaining the required level of connectivity when swarms are large and interactions among robots are intermittent and localized.

Blockchain-enabled robotic systems Swarm robotic systems invalidate most assumptions regarding the maintenance of connectivity or network structure, deteriorating the performance of existing consensus protocols. Strobel et al. (2020) show that when robots communicate on the basis of localized meetings, the protocols seen in Section 2.1.2 progressively deteriorate with the number of Byzantine robots. More critically, if faults stem from adversaries that gain control of some robots, the swarm may become susceptible to devastating attacks such as communication jamming (Multerer et al. 2017), denial-of-service (Xu et al. 2021), and Sybil attacks[2] (Douceur 2002). Strobel et al. (2020) show that even a single Byzantine robot performing such an attack can critically disrupt the collective.

One promising research direction toward achieving secure swarm consensus and coordination is the integration of blockchain technology (Castelló Ferrer 2023, Dorigo et al. 2024†). Robots hosting a blockchain can reach a swarm-wide agreement on the state of a digital database, which can act as a tamper-proof record of robot interactions. This record is an append-only distributed memory that supports auditability of reported events and decisions. Furthermore, it may host a ledger of crypto tokens, which are scarce digital values associated with each robot identity. Combined with public key cryptography, these tokens can be used to mitigate Sybil attacks, double-spend attacks, or spam by imposing a cost on message sending.

[2]In a Sybil attack, a small minority of compromised robots forge many identities to amplify its influence and distort collective decisions.

They can also serve as a basis for a token economy or reputation system, ensuring that as faulty robots accumulate errors, their impact on the system is reduced through reputation mechanisms (Moroncelli et al. 2024†, Pacheco et al. 2024a†, Pacheco et al. 2025†, Strobel et al. 2023†, Van Calck et al. 2023†, Zhao et al. 2025†).

Importantly, blockchains operate even when communication is sparse and robots disseminate messages on a best-effort basis. This means that typically used blockchain consensus protocols are suitable for swarm robotics applications, enabling the creation of a collective database that is resilient to cyberattacks and faulty robots. The next section introduces fundamental knowledge regarding blockchain technology, followed by a review of the most recent literature seeking to integrate robotic and blockchain systems.

2.2 Blockchain technologies

Blockchain technologies aim to reduce the dependence of digital interactions on third-party or centralized intermediaries, such as banks, governments, or large technology companies. This shift is enabled by cryptoeconomic protocols[3] that enable untrusting peers on the internet to create, own shares of, and collectively govern digital resources. This aims to reduce problems such as censorship, data mismanagement, and systemic inefficiencies.

Nakamoto (2008) laid the foundation for this emerging technological field by introducing Bitcoin, the first widely adopted digital currency (now commonly referred to as a cryptocurrency). A central challenge in creating digital currencies is double-spending (Chaum et al. 1990), where users spend the same funds multiple times. Double-spending may occur due to communication delays or malicious activity that leads to conflicting transactions. Without a trusted third party, it was infeasible to determine which of two conflicting transactions should be considered valid. Nakamoto addressed this challenge by using a **blockchain** (Figure 2.3) as a distributed ledger of Bitcoin transactions and devising a novel consensus protocol (Nakamoto consensus, based on proof-of-work—Back 2002, Dwork and Naor 1992) to allow participants to globally agree on the addition of blocks containing transactions to the chain, thereby altering the ledger state. This protocol links cryptographically secured digital actions to real-world energy expenditure, to make malicious behavior economically impractical, as shown in Box 2.3.

[3]Protocols that employ cryptographic primitives and economic mechanisms (e.g., incentives, penalties, and computational or financial constraints) to align participant behavior in open systems without central control.

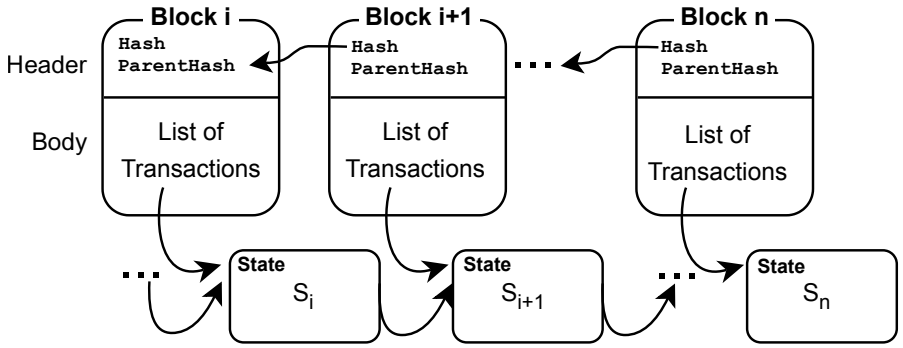


Figure 2.3: Blockchain as a state machine. A blockchain is a linked sequence of blocks. Blocks consist of two sections: a *header* and a *body*. The *header* stores the `ParentHash` of its predecessor. This cryptographic reference ensures immutability: altering any past block would change its hash and break the chain. The *body* contains transactions, which are executed in order to update the state of the ledger. This structure forms a tamper-evident history of state transitions which are replicated by every node in a distributed network.

2.2.1 System organization and design

Public blockchains are designed to be decentralized, transparent, permissionless, and trustless. These are key design aspects that enable blockchain networks to scale and operate globally without censorship.

To achieve this, blockchain networks are composed of thousands of nodes that independently and autonomously validate every transaction and enforce consensus rules to reach agreements on which blocks are added to the blockchain. Different types of nodes with different functions can be deployed. In general, *block producing nodes* (called "miners", "sealers", or "validators", depending on the consensus algorithm) are in charge of aggregating transactions and generating new blocks. *Full nodes* form the basis for security and decentralization by independently checking that every produced block is valid and meets the rules, and by maintaining a local copy of the blockchain. *Light nodes* maintain blockchain headers only which allows them to validate the blockchains' integrity and also check if a given transaction is on-chain by requesting a verifiable receipt to full nodes—without requiring trust and with computation requirements suitable for smart phones and payment terminals. *Users* can interact with the network by sending transactions and maintaining a wallet balance without running a node, but they have to rely on and trust existing nodes to perform most tasks.

Producing blocks is typically an expensive process, requiring some sort of investment, such as hardware, energy, or staked cryptocurrency to protect the network from Sybil attacks and to incentivize honesty. Running full nodes, however,

is *inexpensive by design*,[4] to incentivize network growth and promote security, fault-tolerance, decentralization, and individual sovereignty. Box 2.3 provides a simplified and generalist scheme of a blockchain's consensus and synchronization processes.

Smart contracts Smart contracts, originally proposed by Szabo (Szabo 1997), are contractual clauses encoded in software and enforced through technological means. In blockchain systems, this idea is realized as programs deployed via transactions and executed by the network according to the consensus rules. This makes their execution difficult to censor or unilaterally manipulate once deployed. While Bitcoin supports only basic scripts (for instance, "transfer funds if"), Ethereum (Buterin 2014) is a blockchain developed as a "world computer," capable of executing smart contracts written in a Turing-complete programming language. This technology has the potential to automate finance, governance, and legal processes, creating systems that are more transparent, accessible, and fair.

In blockchains tailored for smart contracts, transactions are not limited to simple transfers of value (e.g., "A sends B 100 coins"). They can also encode contract code deployments (e.g., "A deploys contract X") or function calls to existing contracts (e.g., "B executes function f of contract X with input Y"). Through smart contracts, blockchain networks reach consensus not only on a monetary ledger, but also on the internal state of many smart contracts (e.g., the value of variables). Once a block of transactions is validated and appended to the chain, every participant executes the corresponding contract code, thereby updating and replicating a *global state*. Because smarts contracts are widely used throughout contributions in this thesis, a brief yet encompassing technical presention of Ethereum and its smart contracts is presented in Chapter 4.

2.2.2 Consensus achievement

The blockchain's consensus protocol is a set of rules applied independently by every node to validate blocks and decide whether to add them to their local blockchain. In particular, it establishes which blocks nodes should select when conflicting blocks occur (blocks with the same parent hash). Even if conflicting blocks may happen on occasion (for example, due to network delays), blockchain protocols should ensure convergence to a single canonical chain.

To function over the internet, these protocols must tolerate unreliable and potentially malicious participants, as well as resist cyberattacks and adverse networking conditions. In other words, they must be Byzantine fault-tolerant (BFT,

[4] As an example, it remains common practice to run full Bitcoin nodes using old consumer devices or single-board computers such as a Raspberry Pi (`www.docs.raspiblitz.org/`, accessed October 2025)

Box 2.3: Blockchain consensus and synchronization

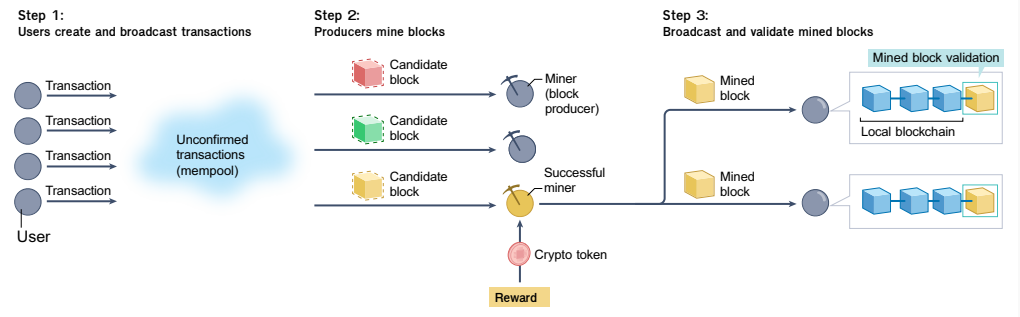


Figure 2.4: Blockchain synchronization steps. Image adapted from Dorigo et al. (2024†).

Step 1: Users broadcast transactions, which are disseminated through a gossiping protocol. Each node maintains a pool of unconfirmed transactions called the mempool (Note: mempools typically differ across nodes).

Step 2: Producers construct candidate blocks[a] containing subsets of their local mempool (typically the transactions offering the highest fees). Afterwards, they broadcast produced blocks to the network

Step 3: Full nodes verify that new blocks follow consensus rules, and validate and execute the transactions they contain. If all is correct, the block is added to the blockchain

- **Conflict resolution**: Sometimes conflicting blocks occur (e.g. due to network latency or two producers finding a valid block nearly simultaneously). Blocks contain a field called `difficulty`, used by consensus protocols to select among diverging blockchain histories (forks).[b]
- **Incentives**: To incentivize producers to operate honestly and remain on the main chain (i.e, with the highest cumulative difficulty), producing blocks yields a *block reward* and transaction fees. Proposing invalid blocks or producing blocks which are not linked to the main chain normally leads to economic loss.

[a] In Nakamoto consensus, candidate blocks must be *mined*, a process where producers spend energy through computational work.

[b] In Nakamoto consensus, the chain with the most accumulated computational work (i.e., difficulty is a probabilistic measure of computational work) is considered the main one. This prevents double-spending attacks, which would require control of 50 % of the network's computational power to outproduce the blocks of honest miners.

Lamport et al. 1982). Although classical BFT consensus algorithms in computer science—such as Byzantine Paxos (Lamport 2011) or Practical BFT (Castro and Liskov 1999, 2002)—can support state replication, they are generally unsuitable for most blockchain applications because they typically:

- assume a fixed set of nodes (processors), making them inherently *permissioned* and dependent on access control layers and administrative oversight;
- rely on timing assumptions of partial synchrony (Dwork et al. 1988), which are unrealistic in large-scale networks with unreliable peers, where message delays can vary unpredictably;
- do not scale to hundreds or thousands of nodes, as communication overhead grows prohibitively with network size n—$O(n^2)$ in synchronous settings and even worse under realistic networking conditions (Civit et al. 2022, Dolev and Reischuk 1985).

For these reasons, the use of classical Byzantine Fault Tolerant (BFT) protocols is generally limited to small, closed consortiums rather than open, global systems. The following paragraphs present the main consensus algorithms employed by **public blockchains**, which allow permissionless participation, as well as by **consortium blockchains**, where participation must be authorized by an administrator or governing committee.

(A) Public/Permissionless blockchains

Nakamoto (2008) was the first to devise a protocol for secure global agreement in a *permissionless* network over the Internet. It relies on peer-to-peer communication where peers exchange messages on a best-effort basis, without requiring that messages are routed to any particular peer. This allows consensus participants to operate on inexpensive or resource-constrained hardware and to exchange messages on a best-effort basis. Unlike classical BFT protocols, which guarantee that agreed-upon state transitions are correct and final, it only provides *probabilistic finality*: the likelihood of a block being reverted decreases exponentially as more blocks are appended to it. This trade-off (weaker finality yet reduced communication and network assumptions) is key to enabling Bitcoin and similar public blockchains to operate as a global network.

A major challenge in permissionless systems is their vulnerability to *Sybil attacks*, where adversaries create multiple identities to increase their influence in the consensus process or to conceal past dishonest behavior. Nakamoto consensus overcomes this through an innovative crypto-economic design that rewards honest participation and resists Sybil manipulation. The core idea is to tie participation in consensus to the expenditure of computational resources via *proof-of-work*.

Nakamoto consensus To produce new blocks (a process known as *mining*), miners must invest in specialized hardware, maintenance, and electricity, while other nodes can check that produced blocks are indeed valid at a minimal cost. This asymmetry between the costs of block production and the ease of validation is fundamental, as it motivates miners to mine honestly and adhere to the protocol; otherwise, their blocks are rejected by the network, leading to economic loss. To reward honest miners, the protocol introduces economic incentives in the form of block rewards, which consist of transaction fees plus an ever-decreasing amount of newly issued tokens. To effectively profit by disrupting the network, an adversary would need to control a majority (more than 50%) of the total computational power to generate a blockchain that grows faster than that of the honest miners. This requirement becomes increasingly impractical as the network size and computation ability of honest miners grows.

One downside of Nakamoto consensus is that while consensus validation is available for everyone, mining and producing blocks is locked behind hardware investment and consumes large quantities of energy (see Box 2.4).

Proof-of-Stake Proposed by King and Nadal (2012), proof-of-stake has become the most prominent alternative to proof-of-work. Many modern blockchains, such as Ethereum (Buterin 2014) and Solana (Yakovenko 2018), employ variations of proof-of-stake integrated with classical BFT protocols to support much higher transaction throughput and mitigate energy consumption. In proof-of-stake systems, consensus nodes (called *validators*) lock up cryptocurrency as a stake. Validators selected to propose blocks risk losing that stake if they violate consensus rules. While proof-of-stake eliminates the need for specialized mining hardware and mitigates energy costs, it also suffers from challenges in achieving fair distribution of crypto tokens. By associating participation in a reward-yielding consensus process with the ownership of tokens, a poorly designed proof-of-stake mechanism can potentially lead to concentration of power, which may weaken the decentralization and censorship resistance that permissionless blockchains aim to preserve (Shifferaw and Lemma 2021). However, Roşu and Saleh (2021) show that this is not always the case, presenting a condition under which token allocations in proof-of-stake stabilize in the long run. Due to its energy efficiency, proof-of-stake has become the de facto industry standard, having evolved into a diverse family of protocols with widely varying designs to prioritize different properties, and tokenomic mechanisms to govern token issuance, distribution, incentives, and penalties (Saleh 2021).

(B) Consortium/permissioned blockchains

Consortium blockchains occupy a middle ground between a public and strictly private blockchain deployment (Dib et al. 2018). In these systems, block production is restricted to predefined group of nodes belonging to organizations (e.g., companies,

Box 2.4: Bitcoin energy consumption

As a curiosity, Figure 2.5 offers context regarding the energy costs of Bitcoin mining. In August 2025, the Bitcoin network's estimated hash rate had reached 1 ZH/s (one Zettahash), meaning miners collectively performed approximately 10^{21} hash computations every second (each hash represents a lottery ticket with a small chance of producing a valid block). An optimistic estimate (assuming all mining is performed by the most efficient miner on the market as of August 2025) yields a total power consumption of 13 GW. More realistic models yield values between 14.5 and 22 GW (De Vries 2018). The current scale and geographical distribution of these mining operations make the Bitcoin network extremely secure and resilient to disruptions or attempts at acquiring more than 50% of the network hashrate to gain control of consensus.

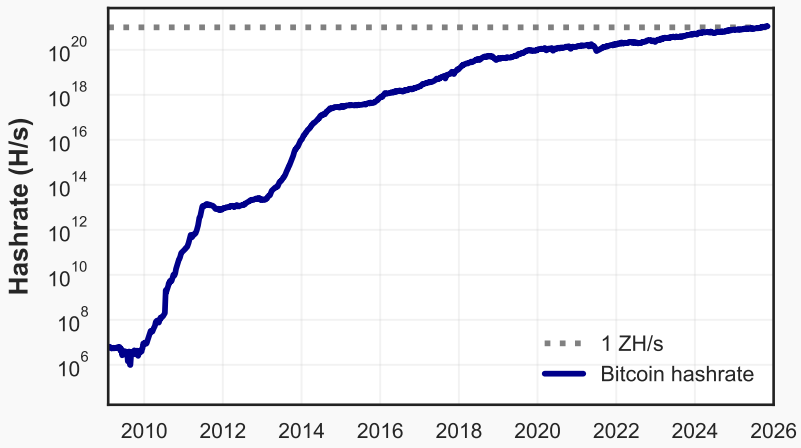


Figure 2.5: Bitcoin hashrate growth over time. Data retrieved from `https://www.blockchain.com/explorer/charts/hash-rate`, November 2025

Other protocols employ modified versions of Nakamoto consensus. For instance, Litecoin, Monero, and Dogecoin use alternative hashing algorithms, thereby forming independent, smaller-scale mining ecosystems. Others propose replacing hashing with useful proof-of-work such as simulating protein folding, searching for large prime numbers, or solving combinatorial optimization problems (Ball et al. 2017, King 2013); or substituting computational power with disk storage, as done with Chia and Filecoin.

governments, banks, or research institutions) that collectively manage the network to achieve a common goal. Participation in consensus is thus *permissioned*, governed by access-control policies or contractual agreements.

In most cases, permissioned blockchains implement variants of classical BFT, enabling *deterministic finality and high transaction throughput*, under the assumption that fewer than one third of participants are inactive or dishonest. Since the number of participants is typically small (tens rather than thousands), the higher communication overhead of deterministic protocols is manageable. Examples include Hyperledger Fabric (Androulaki et al. 2018) and Tendermint (Kwon 2014). Modern protocols such as HotStuff (Yin et al. 2019) and Narwhal&Tusk (Danezis et al. 2022) are able to achieve deterministic consensus while reducing communication costs to linear in the number of participants, significantly improving scalability. If network conditions are unreliable or nodes have limited capabilities (e.g., IoT or robotics networks), permissioned blockchains may employ protocols with probabilistic finality, such as proof-of-authority (Szilágyi 2017) or Snow (Rocket et al. 2019), allowing conflicting blocks to co-exist temporarily while providing resolution mechanisms. In Pacheco et al. (2020[†]) we presented the first physical robot swarm executing proof-of-authority on resource-constrained robots with highly localized and sparse communication.

At first glance, consortium blockchains may appear to undermine some of the core principles of blockchain technology—for instance, decentralization by introducing access controls or administrative oversight. However, they remain valuable in settings where consortium members do not fully trust one another (e.g., if participants are industry competitors, or if some participants may be potential infiltrators who cannot be reliably detected), and where it is desirable to reduce reliance on a central administrator by automating collective decision processes. In particular, in applications with unreliable networking or resource-constrained nodes, permissioned blockchain architectures may offer robustness and scalability beyond what traditional databases can provide.

2.2.3 State of the art

As blockchains become more widely adopted and their application scope expands, developers and innovators face several well-known challenges. This section reviews two such challenges that are directly relevant to this thesis, since they constrain the design space of blockchain-based solutions that can be meaningfully integrated into swarm robotic systems. More broadly, these challenges are widely recognized as key barriers to the adoption of blockchains and smart contracts, as they delineate what is practical and feasible to implement on a blockchain.

This section first reviews the **scalability problem**, focusing on the design tensions between achieving higher transaction throughput and the resource re-

quirements imposed on network nodes. It then introduces and reviews the **oracle problem**, which concerns the challenge of introducing real-world information to blockchains without relying on trusted information providers.

(A) Scalability problem

Previous sections highlighted how Bitcoin maintains a global decentralized network due to its design that ensures running a full node is affordable for individuals. However, this is achieved by limiting the number of transactions that can be processed—otherwise, the storage, processing, and communication requirements on nodes would increase accordingly. To manage the costs of running a Bitcoin node, Bitcoin blocks are limited in size (transaction payload) and frequency.[5] Similarly, Ethereum uses a measure of the computation cost of executing smart contracts (called "gas") to impose per-block computation limits.

Increasing throughput is possible by modifying either block limits or frequency, which in turn would yield higher demands on nodes.[6] As a result, scalability in *public* blockchains is constrained by the need to keep the bandwidth, computation, and storage requirements sufficiently low to preserve permissionless participation and decentralization.

State-of-the-art scaling approaches tend to follow two main paradigms: (i) offloading computation and storage to off-chain components through layered designs, or (ii) adopting high-throughput consensus protocols at the base layer, at the cost of higher hardware requirements. Consortium blockchains can pursue alternative designs, as they relax the requirement for open participation and can instead be tailored to specific applications and node capabilities.

The following paragraphs present the scaling strategies in use by the three most adopted blockchains—Bitcoin, Ethereum and Solana—and also discuss permissioned designs. Croman et al. (2016) provides a more comprehensive discussion on scaling paths for blockchain systems. Monte et al. (2020) discuss the *scalability trilemma* which highlights the design tensions between three important properties of a blockchain: *decentralization* (i.e., the costs of running nodes), *scalability* (i.e., transaction throughput) and *security* (i.e., consensus resilience to attackers).

The Lightning Network Bitcoin's conservative design emphasizes decentralization and security at the base layer. In order to meet the global demand for financial transactions, off-chain payment layers are supported. In this scheme, the blockchain serves as a settlement rail while most day-to-day payments occur through off-chain exchanges. This illustrates the core logic of off-chain scaling: *moving high-frequency*

[5]The Bitcoin network achieves 3 to 7 transactions per second under ideal conditions, as of November 2025 (1–2 MB blocks, occurring every 10 minutes on average).

[6]The Bitcoin Cash community, in 2017, increased Bitcoin's original maximum block size. This led to a *hard fork*, in which a blockchain splits into two others that follow different rules.

interactions off the global ledger while retaining the blockchain as a settlement layer. Poon and Dryja (2016) present the most prominent instantiation of this model, called Bitcoin's *Lightning Network*, briefly explained in Box 2.5. In Lightning, users open payment channels to one another by locking up Bitcoin in a mutual contract. Funds can then be routed between users so that a user can send payments even without a direct channel to the recipient. However, optimally routing payments is a challenge that depends on the availability of liquidity along paths (Di Stasi et al. 2018). Grunspan and Pérez-Marco (2018) propose an approach inspired by ant path-finding—which, however, the authors distinguish from Ant Colony Optimization (Dorigo and Stützle 2010). Empirical studies investigate how routing tends to concentrate in subsets of nodes, creating potential liquidity bottlenecks and censorship risks (Zabka et al. 2023).

Layered blockchain design Smart contract blockchains face broader scalability challenges than payment-focused ones due to the computational costs of executing arbitrary computer code. State-of-the-art scaling strategies execute smart contracts or store data externally layers while periodically posting succinct commitments to the blockchain, which serves as a secure settlement and finality layer.

Buterin (2024) presented Ethereum's *rollup-centric design*, in which rollup nodes batch many user transactions, execute them off-chain, and post the resulting state to a smart contract hosted on the base layer. Under this design, transaction throughput increases substantially, but security depends on (i) proving correct off-chain execution, and (ii) maintaining data availability.

Two dominant rollup families differ in how they ensure *correctness* of contract execution:

- **Optimistic rollups** assume executions are correct by default, but allow *fraud proofs* during a challenge period, which delays finality.
- **ZK rollups** post zero-knowledge proofs that executions are correct, enabling faster finality but requiring dedicated nodes to compute proofs.

A key bottleneck for rollups is *data availability*: Nodes need access to enough transaction data to independently reconstruct the rollup's state. If that data is controlled by a single operator, it could be lost or the operator could censor users. To mitigate storage costs, Ethereum's rollup data is kept only for a limited period, sufficient for validators to verify the state (Buterin et al. 2022).

High-throughput consensus Solana, specified by Yakovenko (2018), represents an alternative point in the blockchain design space: instead of pushing for off-chain execution, it pursues a comparatively monolithic architecture that aims to achieve high throughput directly on the base layer by optimizing its consensus protocol, networking, and parallel execution. Unlike classic BFT protocols, which require

Box 2.5: Bitcoin's Lightning Network

Lightning is a network of *bidirectional payment channels* (Figure 2.6). Channels between two peers are opened by publishing an *on-chain* funding transaction that locks a specified amount of Bitcoin, which is then bound to the channel. The two peers can then privately exchange *off-chain* transactions (signed by both parties), which rebalance the channel balances held by each peer. To close a channel and settle final balances on the Bitcoin ledger, a second *on-chain* transaction is required. In this way, any number of transactions between two peers can be compressed using two on-chain transactions, enabling privacy and near-instant finality.

To scale beyond pairwise channels, Lightning supports *multi-hop payments* (Figure 2.6). Payments are routed across the network, without the need to open a direct channel between the sender and receiver. Each channel along the route is rebalanced, ensuring that funds arrive at the receiver. To achieve this, Lightning relies on a gossip-based routing layer and a standardized protocol specified online (Lightning developers 2025).

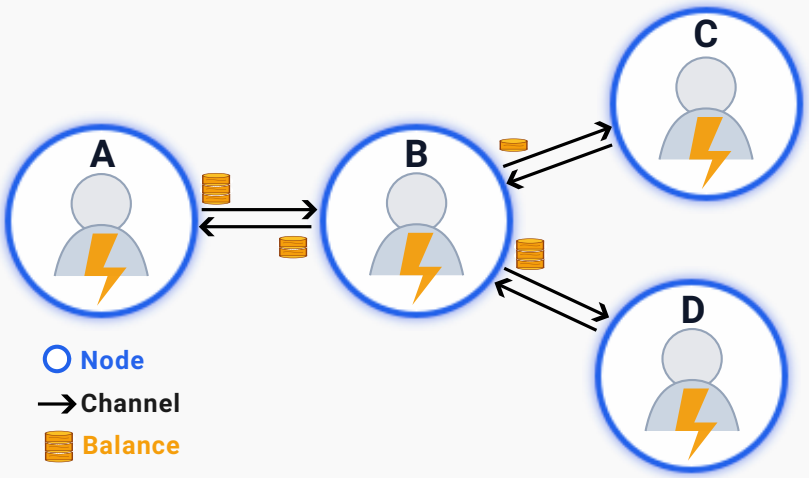


Figure 2.6: Lightning peer-to-peer payment network. In the pictured example, A can send direct payments to B or route payments to C and D through B. The channel A–B has a total balance of 5 tokens, with 3 tokens on A's side and 2 on B's side. Each transaction between A and B shifts the balance between the two sides. A can send at most 3 tokens directly to B, but routed payments are constrained by the balance in B's channels: A can send only 1 token to C or 3 tokens to D.

multiple communication rounds to agree on and finalize transaction ordering, Solana introduces *Proof of History (PoH)* to provide a verifiable common reference for ordering and timing. Validators then run a voting protocol based on proof-of-stake called Tower BFT, using PoH as a network clock to order events. This approach reduces communication overhead and confirmation latency.

However, Solana has higher requirements and performance expectations for its validators in terms of network bandwidth, CPU, memory, and operational reliability. In other words, Solana achieves high throughput and low latency at the base layer while simultaneously increasing consensus entry barriers, shifting node hosting to professional operators. Despite this, due to its open protocols, lack of access control layers, and wide variety of software implementations, Solana remains widely recognized as a public blockchain (Pierro and Tonelli 2022).

Application-driven design (permissioned) In permissioned blockchains, accessible participation is not a strict requirement. For instance, in a *consortium* blockchain for supply chain management, the set of nodes consists of a relatively small group of organizations that maintain nodes on professional hardware. As a result, these systems can adopt higher-performance base layer consensus, with faster finality, richer privacy controls, and application-specific throughput targets. In general, this comes at the cost of weaker censorship resistance and decentralization guarantees compared to public chains.

This thesis explores the use of *permissioned* blockchains for swarm robotics applications. Since robots have limited hardware and communication capabilities, deploying a blockchain that scales with swarm size also yields design tensions between transaction throughput, hardware demands on individual robots, and the degree of decentralization. As such, the scalability of blockchains maintained by robots will be discussed throughout this thesis. In particular, Chapter 3 discusses the scalability challenges of a blockchain deployment in a robot swarm, and Chapter 5 presents scalability results in swarms of up to 24 real robots and 120 simulated robots.

(B) Oracle problem

In a blockchain, every full node must be able to deterministically execute transactions to replicate an identical global state. This design limits the capabilities of smart contracts: they cannot natively use randomness or stochastic procedures, and they can only access external information (i.e., information not contained within the blockchain itself) through the contents transactions (Antonopoulos and Wood 2018). For many applications, the algorithmic decision-making that is enabled by smart contracts requires knowledge of external information such as asset prices, weather events, sensor readings, and logistics milestones. To meet these

requirements, *blockchain oracles* are users, mechanisms, or services that deliver external data to smart contracts via transactions.

The *oracle problem* is the challenge of providing data to a contract in a way that is both trustworthy and decentralized: if a contract's state transitions depend on an oracle data feed, then whoever controls the oracle can manipulate outcomes. Relying on a single oracle can introduce a single point of failure, corruption, censorship, or attack that can irreversibly trigger incorrect executions. The oracle problem is fundamentally a trust problem: how to provide reliable real-world inputs to smart contracts without reintroducing a trusted intermediary that undermines the security and decentralization guarantees of the blockchain.

This thesis considers the potential use of blockchains to coordinate robot swarms, raising the oracle problem since unreliable data from the robots must be provided to the blockchain for decision-making. Chapter 3 presents the oracle problem in the context of robotics applications, and Chapter 6 presents Swarm Oracle, the main contribution of this thesis. The remainder of this section reviews common approaches to the oracle problem and the current open challenges.

Centralized and committee-based oracles The simplest approach is to designate a single trusted publisher, or a small committee, to post data on-chain. Such architectures can be operationally efficient and may be appropriate when participants already trust the data provider, for example, an environmental agency in a consortium blockchain that manages natural resources. However, they contradict the trustless design goal of public blockchains: correctness becomes contingent on the honesty of a small set of entities, and the oracle becomes a censorable bottleneck (Caldarelli 2020). Consequently, committee-based designs are best understood as a pragmatic engineering compromise, not a general solution.

Optimistic and dispute-based oracles The design of early oracles on public blockchains relied on optimistic acceptance with dispute resolution. In these protocols, a publisher posts a claim on-chain together with a stake; the claim is treated as valid unless challenged within a dispute window. If challenged, a resolution mechanism based on peer voting or arbitration determines the outcome and reallocates bonds, penalizing dishonest behavior. The UMA Optimistic Oracle (2025) is a representative example of this paradigm. Edgar (2017) proposed an early example of a crowdsourcing oracle system that relied on economic incentives to have app users answer arbitrary queries posed by contracts. Lesaege et al. (2019) introduced human mediators to resolve disputed claims, for example, those with an ambiguous or subjective nature.

Optimistic oracles were initially attractive because they could address arbitrary questions (e.g., "Who is the current president of the USA?"), during a time when demand for blockchain oracles was not yet established. However, they require a reliable network of publishers, challengers, and jurors in order to benefit from the

wisdom of the crowd effect and security. Additionally, the lack of automation and inability to provide timely and guaranteed outcomes limited their applications as blockchain technology advanced.

Decentralized oracle networks A dominant approach in public blockchain smart contracts is to use a *decentralized oracle network* (DON), where multiple independently operated oracle nodes automatically fetch data off-chain and collectively produce an on-chain report. Different oracle projects differentiate on data types, latency, trust assumptions, and on the blockchain ecosystems or application classes they target.

Chainlink is one of the first and most successful DONs. In its canonical design (Ellis et al. 2017), a requesting contract specifies a data query; a set of software oracle nodes provides data obtained from online sources or APIs to the blockchain; and an on-chain aggregation contract computes an averaged final value that is then consumed by decision-making smart contracts. To reduce costs, reporting and aggregation has since been moved off-chain (Breidenbach et al. 2021). API3 (Benligiray et al. 2020) advocates that sources or data issuers operate oracle infrastructure themselves, aiming to improve provenance guarantees by reducing intermediary layers (e.g., by having news channels host oracle nodes instead of relying on software to fetch online news). Zhang et al. (2016) introduced Town Crier, which uses trusted execution environments (Intel SGX) to provide verifiable attestations that specific HTTPS content was fetched from a given website, limiting the oracle operator's ability to tamper with or falsify the data. Tellor (2023) is a permissionless reporting approach in which reporters submit values under a cryptoeconomic incentive model with disputes, favoring openness and censorship resistance over curated feed operation. Band Protocol (2024) aims to bridge information across different blockchains (the information on one blockchain is external to another), emphasizing determinism, correctness and low-latency in its design.

Hardware oracles Most research on DONs focuses on software oracles that retrieve data from online sources. By contrast, *hardware oracles* use physical sensors to capture data directly from the environment. In practice, the term "hardware oracle" is not widely used, partly because demand for sensor-driven data in smart contracts is still limited. However, many decentralized physical infrastructure networks (DePINs)[7] display similar functionality. These DePINs consist of distributed networks of user devices that receive crypto token rewards in exchange for sensor data (Lin et al. 2025).

Haleem et al. (2024) introduced Helium, a distributed IoT network that provides wireless coverage while gathering geolocation and connectivity data for on-chain ap-

[7]DePINs are further discussed in Chapter 3.

plications. Auki (2025) introduced the "posemesh", a shared digital reconstruction of local spatial context from consumer cameras (e.g., from user phones or wearable devices) to support robotics and other applications requiring spatial awareness. Horton et al. (2023) provides geodetic reference data through a network of affordable GNSS reference stations that emit positioning and atmospheric observation data. WeatherXM (2025) distributes tokens to incentivize deployment of weather stations to create localized and high density meteorological feeds. Silencio Network (2025) turns smartphones into distributed noise sensors to construct high-resolution noise maps. Hivemapper (2025) crowdsources road-level imagery and geospatial intelligence from dashboard cameras users can install in their vehicles.

Open challenges The oracle problem remains largely unsolved. A central challenge lies in ensuring that DONs can withstand coordinated attempts to manipulate data feeds. Data quality poses an additional difficulty, as many oracle nodes rely on third parties rather than directly observing data sources. As a result, DONs are vulnerable to correlated failures, for example when multiple oracles depend on the same data providers, services, or potentially biased software implementations. Existing approaches motivate primary sources to host their own oracle nodes, employing economic incentives to enhance accountability and improve data quality (Breidenbach et al. 2021).

Another challenge is the limited scope of existing oracles. Autonomous software oracles are restricted to data that is available online or via APIs, while hardware oracles are typically static, collect specific data types, and cannot respond to arbitrary queries. Furthermore, hardware oracles such as IoT devices and physical sensors are affected by environmental noise, and the devices themselves can be tampered with or compromised. Through Swarm Oracle, Chapter 6 addresses some of these challenges by employing a robot swarm as a mobile and flexible DON. Individual robots collect observations using onboard sensors and reach collective agreement through an on-chain BFT protocol. This design ensures correctness in the presence of attackers and promotes unbiased outcomes by leveraging heterogeneity and competition among robots operated by different stakeholders.

2.3 Blockchain-based robotics

Research on integrating blockchain technology with robotics has expanded rapidly in recent years, as illustrated in Figure 2.7. In swarm robotics, Strobel et al. (2018) published pioneering work in which robots collectively maintain a blockchain ledger. However, most existing work treats robots as users of externally hosted blockchains rather than as ledger maintainers. The following paragraphs present in detail distinct system architectures for integrating blockchain with robotic systems.

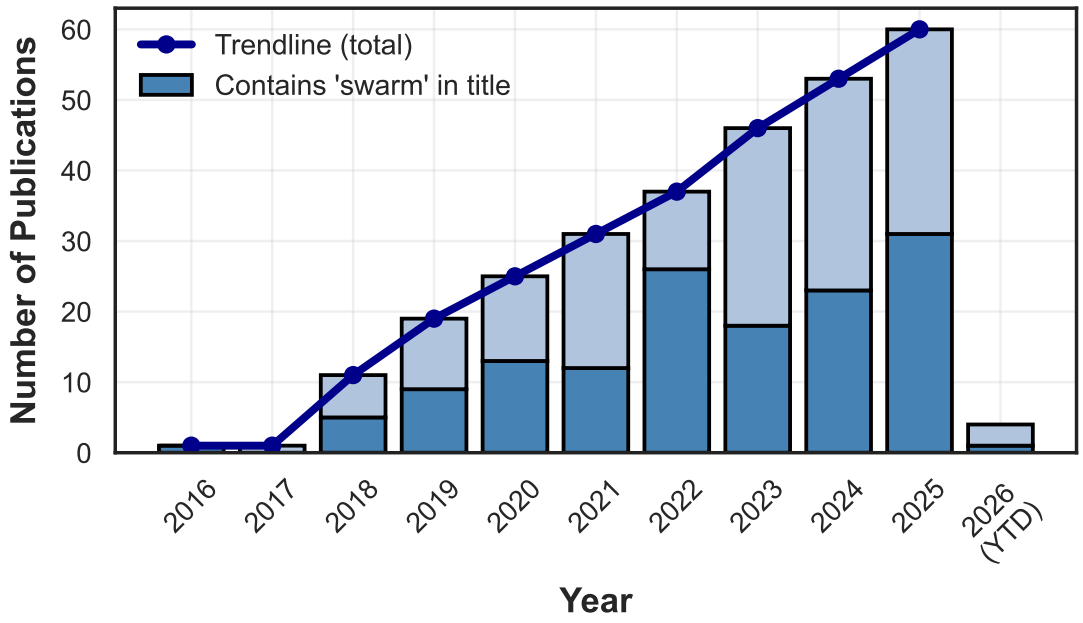


Figure 2.7: Publications integrating blockchain and robotics. Yearly counts (non-cumulative) of Google Scholar records whose title contains at least one robotics keyword ("robot" or "swarm") and at least one blockchain keyword ("blockchain", "distributed ledger" or "smart contract"). The data shown was retrieved on 30-01-2026. Milestones: Castelló Ferrer (2016) launches the idea of integrating blockchain with swarm robotics; Strobel et al. (2018) proposes blockchain for securing consensus in swarms; Pacheco et al. (2020†) presents the first physical robot swarm hosting a blockchain; and Aditya et al. (2021), Castelló Ferrer (2023), Dorigo et al. (2024†), and Peña Queralta et al. (2023) provide perspective reviews on the emerging topic.

2.3.1 System architectures

One can construct systems where the robots are *nodes*, maintaining the blockchain themselves, and systems where the robots are *users*, interacting with an externally hosted blockchain, as shown in Figure 2.8. Which architecture is most suitable depends on factors such as the robots' hardware and capabilities, the desired level of autonomy, and the deployment scenario (e.g., remote locations or city environments).

Architecture 1 achieves full autonomy by having the robots maintain the blockchain themselves. Each robot participates in consensus by casting transactions, generating and validating blocks, and propagating them throughout the swarm. All of the experimental studies in this thesis follow this architecture.

Architecture 2 allows robots to leverage the security, financial settlement, and human interactions (to exchange services and monetary value) of an external blockchain, without incurring the costs of hosting a blockchain themselves. However, this architecture requires that at least some of the robots can reliably connect to blockchain nodes; otherwise the system reverts to blockchain-free operation.

Architecture 3 is a hybrid architecture where the robots autonomously host an internal blockchain, while periodically synchronizing information (e.g., providing operational data and receiving new tasks) with an external blockchain. In this case, the robots can benefit from the availability of the blockchain without relying on frequent connections to the external one. In certain deployment scenarios, Swarm Oracle can be seen as an instantiation of this architecture (Pacheco et al. 2025†).

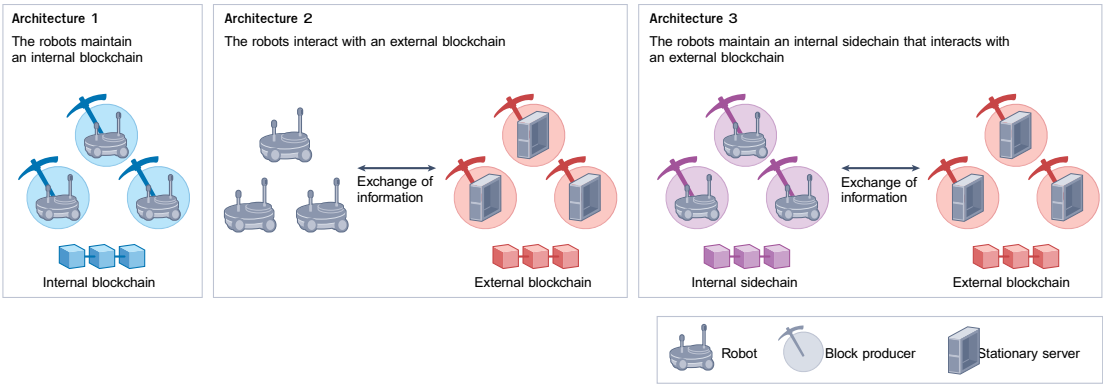


Figure 2.8: Architectures for the integration of blockchain and robotics. Image adapted from Dorigo et al. (2024†).

2.3.2 State of the art

The following paragraphs review research that integrates blockchain and robotic systems under the proposed architectures. The discussion is limited to Architecture 1 and Architecture 2, as no existing work addresses Architecture 3—with the exception of Swarm Oracle (Pacheco et al. 2025†), which is presented in Chapter 6.

(A) Architecture 1

Most work on blockchain-enabled robot swarms studies the case which the robots themselves host the blockchain (see Figure 2.8, Architecture 1). Initially, this line of work was motivated by the insight that security issues in robot swarms could be improved through the use of blockchain-based smart contracts.

Strobel et al. (2018) experimentally demonstrated how simulated robots in a swarm can be controlled via smart contracts. Building on this, they showed how blockchain technology could add a security layer on top of existing swarm robotics algorithms in a binary decision-making scenario. The smart contract enabled the robot swarm to detect inconsistencies in robots' behaviors in a robot swarm where some of the robots were Byzantine. Strobel and Dorigo (2018) extended this work to the collective estimation of an environmental feature. They showed that a reputation management system could be implemented in a smart contract by assigning a trust value to robots in a swarm, over time neutralizing the misleading estimates of Byzantine robots. Strobel et al. (2020) compared the performance of a robot swarm controlled by a blockchain-based smart contract with that of robot swarms using consensus protocols available in the literature (presented in Section 2.1.2). Results showed that blockchain-based smart contracts were capable of identifying and excluding Byzantine robots, whereas the existing consensus protocols failed. Importantly, by leveraging blockchains' public-key cryptography and permissioned protocols, they were also resilient to Sybil attacks (Douceur 2002).

As an alternative to blockchains, researchers have also deployed other types of distributed ledgers (such as directed acyclic graphs: Keramat et al. 2023, Salimpour et al. 2023, Tran et al. 2019) on robot swarms. While the goal is the same—that is, to allow robots to synchronize a secure and global database without relying on external infrastructure—the two designs provide different features in terms of partition-tolerance, finality, and costs.

The contributions included in this thesis, as well as other co-authored works, fall inline with Architecture 1. In Pacheco et al. (2020†) we addressed the question of whether real robots' computing power, random-access memory, and storage capabilities could handle the requirements imposed by blockchain technology; and in Strobel et al. (2023†) we extended the study to much larger swarms and sparser

communication networks. We showed that secure consensus could still be achieved under realistic communication constraints, and that the costs associated with maintaining the blockchain remained manageable as the swarm size increased.

In Zhao et al. (2023†), we shifted the focus from the proof-of-concept consensus task to propose a modular and generic smart contract that can be adjusted to different tasks by modifying the smart contract's aggregation function, which specifies how values from individual robots are merged. Then, through a Byzantine fault-tolerant voting game the swarm reaches a final, global decision on whether the aggregated values are correct—even when up to one-third of the robots are Byzantine. This ability to reach Byzantine fault-tolerant agreements at the swarm level was then formalized and tested in extensive experimental studies in Pacheco et al. (2025†) to form Swarm Oracle. In this work, the swarm forms an autonomous decentralized oracle network that can flexibly provide trustworthy information to smart contracts to support decision-making processes. These decision processes can be hosted in external blockchains, but also within the swarm itself—forming the basis for a novel form of self-organization built upon secure blockchain consensus.

In several works, we applied these principles to concrete swarm application scenarios. In Pacheco et al. (2022†), we showed how a smart contract can supervise a foraging task by aggregating information from individual robots and assigning them to resources in a coordinated manner, thereby reducing interference and improving overall efficiency. In Gupta et al. (2024†), we showed how swarm-wide behavioral changes can be reliably triggered in response to critical events by exploiting blockchain's global consensus. In Pacheco et al. (2024a†), we demonstrated how robots can collaboratively build a shared neural network model by aggregating models trained locally by each robot, while preventing faulty robots from degrading the shared model. Similarly, in Moroncelli et al. (2024†), we addressed the challenge of building a consistent global map by using a smart contract to enforce geometric constraints between robots' trajectories and coordinate transformations which, if incorrect, would deteriorate the map optimization and lead to high costs.

In Van Calck et al. (2023†), we explored how swarms composed of profit-seeking robots can be indirectly coordinated by reward and penalty mechanisms, demonstrating that appropriate incentives can encourage cooperation when agents are greedy and discourage harmful behavior from Byzantine robots. In Zhao et al. (2025†), Swarm Oracle was applied to a task allocation framework that accounts for heterogeneous robot capabilities, ensuring fair participation and equitable outcomes while limiting the impact of faulty robots.

(B) Architecture 2

Instead of hosting the blockchain themselves, robots can also connect to an external blockchain to upload data or to receive instructions, software updates or tasks.

There are several examples of multi-robot systems that use an external blockchain to coordinate the robots' activities. Castelló Ferrer et al. (2021b) show that Byzantine robots in a leader-follower formation can temporarily misguide their peers, but that this misguidance can eventually be detected and undone by analyzing the transaction history on the external blockchain. Alsamhi et al. (2023) outline the potential of blockchain technology to enhance the environmental exploration capabilities of swarms consisting of unmanned aerial vehicles (UAVs). In particular, the framework they propose improves data integrity and protection against malicious attacks and increases the energy efficiency of the swarm of drones by exploiting an external blockchain. Mokhtar et al. (2019) use a smart contract to coordinate multi-robot path planning: Each robot stores its planned paths on the blockchain, allowing them to detect if their path would lead to collisions with other robots and adapt their path accordingly. Santos De Campos et al. (2021) provide an initial proof of concept for surveying points-of-interest by multiple UAVs using a smart contract deployed on IOTA[8] to send rewards to robots for surveying locations. Similarly, Grey et al. (2020) show how to incentivize the completion of tasks through smart contracts that assign different roles to robots to perform tasks. This work was then extended by showing how robots in a heterogeneous multi-robot system can allocate tasks (e.g., object retrieval and object transportation) in a warehouse application using smart contracts (Grey et al. 2021, Mallikarachchi et al. 2022).

Blockchains can also provide novel forms of human-robot interaction. Castelló Ferrer et al. (2018), for example, proposes a blockchain to store personal data of patients and protect their privacy when a robot needs access to the data. Alsamhi and Lee (2021) propose a framework based on the combination of blockchain and multi-robot systems for battling pandemics. The authors present different use cases, such as spotting lockdown violations or delivering medication by multi-robot systems, and discuss how blockchain technology could be used to integrate information from different parties (for example, through sharing medical information on a blockchain among different healthcare providers).

Similar to a flight recorder (also referred to as a "black box"), a blockchain can also be used as a secure event registration mechanism and create tamper-proof logs (White et al. 2019). If part of the robots are destroyed or lost, or even in the extreme case in which only a single robot can be retrieved, it could be possible to recover the events of a mission as each robot keeps a copy of the blockchain. Event logging is, for example, done in a project called RobotChain (Lopes and Alexandre 2019, Lopes et al. 2021). Their framework also enables regular snapshots of the blockchain state, reducing the data storage requirements on the individual robots.

Researchers have also studied the intersection between economics and robotics,

[8] `www.iota.org` is a distributed ledger which employs a directed acyclic graph in its architecture, originally developed for the Internet of Things

called "robonomics". Several conceptual papers and initial proofs of concept show how a blockchain-based smart contract residing on an external blockchain can interface with a robot, building the basis for human-to-robot economic transactions and robots-as-a-service applications (Kapitonov et al. 2021). Kapitonov et al. (2017) describe the Autonomous Intelligent Robot Agent (AIRA) project, a proof of concept software that exploits smart contracts on an external Ethereum blockchain to "hire" multi-robot systems composed of UAVs. Gaka-Chu Castelló Ferrer et al. (2023) is an experimental robotic art project in which an economically autonomous robot maintains its own Ethereum wallet, creates physical paintings, and sells them via blockchain-based online auctions.

2.4 Chapter summary

Overall, this chapter introduced the key concepts and background knowledge required for the remainder of the thesis and situated the work within the state of the art.

First, the chapter introduced the two foundational fields underpinning this thesis: *swarm robotics and blockchain technology*. It highlighted the commonalities across the two fields (e.g., decentralization, redundancy, self-organization and probabilistic consensus achievement) and their fundamental differences (e.g., physical embodiment, behavior and fault models, communication and hardware assumptions). Swarm systems rely on cooperation and physical interactions to achieve scalable collective behaviors, whereas blockchain networks must contend with Byzantine participants in open networks. These contrasting assumptions lead to slightly distinct notions of consensus, fault tolerance, and scalability.

Afterwards, the chapter outlined—for each field—the organizational principles and approaches to the design of decentralized systems, in particular, their methods for consensus achievement. It then reviewed the state of the art on the key challenges related to this thesis' contributions. In robotics, the review covered fault-tolerance and robustness, particularly when Byzantine or adversarial agents are present; and in blockchain technology, it covered the scalability and oracle problems.

Finally, the chapter outlined three system architectures for the integration of blockchain and multi-robot systems and reviewed the literature following each architecture. This thesis focuses on architectures where robots maintain the blockchain themselves, enabling autonomous and resilient coordination without external infrastructure.

Chapter 3

Perspective

This chapter presents a unifying perspective on the integration of blockchain technology and swarm robotics that draws on nine years of research at IRIDIA, as well as broader advances in blockchain-based multi-robot systems.

This perspective is structured around the *challenges* and *opportunities* that arise when integrating blockchain technology with swarm robotics. Section 3.1 introduces and motivates this perspective, situating it within the broader landscape of robotics and blockchain research. Section 3.2 discusses three central challenges: hardware feasibility, scalability, and the oracle problem. Section 3.3 examines five opportunities enabled by this integration: self-governance; economic interactions; security, trust, accountability, compliance; data consistency; and top-down swarm design.

In the chapters' summary, we draw a structural link between this perspective and the thesis contributions, outlining how the chapters that follow tackle the identified challenges and enable some of the opportunities.

3.1 Introduction

Robot swarms can monitor and cover large-scale areas, mitigate single points of failure, replace broken teammates, and reconfigure their shape to match the demands of different tasks (Dorigo et al. 2014, Mathews et al. 2019, Timmis et al. 2016). Despite this potential, most demonstrations of robot swarms remain confined to controlled research environments (Dorigo et al. 2021), and when multiple mobile robots are deployed beyond the lab, they are typically operated in safe, structured indoor settings and often depend on external infrastructure for coordination and monitoring (Fragapane et al. 2021).

As hardware, control, and economic constraints are gradually overcome, swarm-robotics applications are expected to become increasingly common (Dorigo et al.

2020, Yang et al. 2018). However, coordinating and securing interactions in large robot collectives operating in outdoor and unstructured environments remains a critical challenge. To protect robots, their owners, the environment, and the humans they interact with, swarms should provide behavioral guarantees and accountability at both individual and collective levels (Wilson et al. 2023). Such guarantees are difficult to obtain in swarm robotics because behaviors emerging from bottom-up design and local interactions rarely come with explicit assurances of safety or performance (Hunt and Hauert 2020).

This perspective discusses how blockchain technology can potentially strengthen the security and coordination capabilities of robot swarms, while highlighting open problems and promising research directions. It draws on nearly a decade of work on the topic, which gained prominence around 2018 (see Figure 2.7 in Chapter 2). At IRIDIA, this line of research began with Strobel et al.'s (2018) early contribution and continued through works presented in this thesis, initiated in 2019. Accordingly, the perspective reflects an informed view shaped by several years of research at the intersection of blockchain technology and swarm robotics. It is complemented by perspectives from adjacent communities working at related intersections, including robotics (Aditya et al. 2021), multi-robot systems (Peña Queralta et al. 2023), artificial intelligence (Salah et al. 2019), and edge computing (Peña Queralta and Westerlund 2021). Complementary discussions on the use of blockchains for securing robot swarms are also provided by Castelló Ferrer (2023).

3.2 Challenges

Blockchains were originally engineered for networks of stationary computers connected via the Internet. On the one hand, several core assumptions align with swarm robotics—namely asynchronous communication, unreliable peers, and the possible presence of unidentified malicious agents. On the other hand, in a swarm, network agents are mobile, communicate over local and often intermittent links, and operate under limited compute, memory, and energy budgets. These characteristics introduce new challenges for integrating blockchains into robot swarms.

We identify three central challenges that must be addressed to make blockchain-enabled robot swarms practical: (i) *managing hardware requirements* on resource-constrained robots; (ii) *achieving scalability* so that latency, bandwidth, storage, and computational demands remain feasible as the swarm size and message volume grow; and (iii) addressing the *oracle problem*, i.e., the gap between blockchain's digital interactions and robots' physical interactions.

3.2.1 Hardware feasibility

Public blockchains are designed so that *consensus validation is inexpensive*: any node can independently validate the blockchain on modest hardware and with limited connectivity, even as the network scales.[1] This design makes many protocols suitable for robotic platforms, as discussed in Chapter 2. However, consensus validation does not equate to consensus participation through block production. Producing blocks in Bitcoin requires specialized hardware and substantial energy expenditure, and similar barriers to entry are common across most globally available public blockchains as a form of Sybil protection.

Existing mechanisms for admitting new consensus participants often lead to security risks, and proof-of-work-based protocols are vulnerable to attacks in which the work is performed by more capable machines impersonating as robots. Therefore the design of admission and participation rules that robust to Sybil attacks remains an open challenge in robotics. Several attempts have been made, however lack tests on real robots. Lokhava et al. (2019) proposes a consensus mechanism based on social trust graphs which, in a robotics context, could be associated with robots peer-driven reputation scores. Wardega et al. (2023) propose the Decentralized Blocklist Protocol, which allows robots to formally accuse misbehaving peers, leading to a loss of trust and privileges for offenders, as demonstrated in simulations with up to 100 robots. Another, yet unexplored, alternative is the concept of a proof-of-physical-work protocol, where the right to append to the blockchain is earned by first completing a useful task, such as transporting an object, performing a computation and effectively relaying messages among peers (Hoffmann 2022, Strobel et al. 2018). The key challenge here is in the creation of scalable, secure, and verifiable methods to ensure "work" is valid.

Due to these obstacles, research on blockchain-enabled robotics has converged on two deployment patterns: (i) using external blockchains; or (ii) running permissioned consensus, in which robots are authenticated nodes that produce and validate blocks. These patterns correspond, respectively, to the Architectures 2 and 1, reviewed in Chapter 2. In swarms, permissioned blockchains maintained autonomously by the robots are most common as they offer the following properties:

- Independence from external infrastructure.
- Resilience to loss of robots or communication network links.
- The blockchain functionalities can be tailored to application requirements.
- New robots can join and perform tasks without participating in consensus.
- New consensus nodes can be added by administrators.

[1] As a concrete example, running a full Bitcoin node is feasible on a Raspberry Pi with a standard home Internet connection (`www.docs.raspiblitz.org/`, accessed February 2025).

Through a permissioned blockchain, the swarm inherits all of blockchain's security and smart contract functionalities, while circumventing the high energetic costs of proof-of-work. In Chapter 5 we deploy a private Ethereum network using proof-of-authority consensus on resource-constrained robots, scaling from an initial 10 physical robots (Pacheco et al. 2020†) to swarms of 24 physical and 120 simulated robots (Strobel et al. 2023†). Other authors have developed similar setups; for instance, Salimi et al. (2023) demonstrated the use of the Hyperledger Fabric framework, a permissioned consensus based on leader election called Raft (Ongaro and Ousterhout 2014).

3.2.2 Scalability

Swarms consisting of minimalist or small robotic platforms (Connell 1990) may be *memoryless*: robots keep no record of past interactions and react purely to locally available information (Flocchini and Prencipe 2012). Most modern swarm applications, however, assume a small local memory or "opinion" that results from repeated interactions with the environment and peers (Valentini et al. 2017). In either case, the size of robots' internal memory state does not grow with time or with the number of robots.

A blockchain-based swarm deviates from this design principle, since each robot maintains a local blockchain that serves as a historical log of inputs contributed by swarm members. Because swarm hardware generally offers far less computation, storage, and communication capacity than machines in conventional blockchain networks, this raises important questions about how such systems scale as swarm size increases.

In general, three key design principles help ensure that system requirements remain within the capabilities of modern large-scale robot swarms:

1. Storage costs should scale at most linearly with time and number of robots.

2. Information exchange should remain local and gossip-based, ensuring that no individual robot is burdened with excessive communication or computation over short time horizons.

3. All employed algorithms (including those executed within smart contracts) should exhibit at most linear computational complexity as the number of robots and task-related variables (e.g., tasks, roles, or assignments) increases.

In the following, we analyze the costs typically incurred by a blockchain-based deployment in terms of the growth in **storage**, **communication**, and **computation**, and discuss techniques for mitigating these costs.

(A) Storage growth

Blocks are typically generated at predefined intervals and have a maximum data capacity. For this reason, blockchain storage requirements grow predictably. While it is normally required that nodes maintain a local copy of the blockchain, several cost-mitigation strategies are built into blockchain protocols and can be leveraged for swarm robotics deployments.

State maintenance Instead of storing historical ledger states, each full node maintains only a current state, which is obtained by executing every transaction on the chain. When some specific historic state snapshots are needed, methods such as Ethereum's archive nodes can be implemented (Ethereum Foundation 2025), thereby offloading data storage to external infrastructure instead of the robots.

Pruning Once historic blocks have been validated, nodes can discard the transactions they contain, maintaining only the current state, block headers, and a sliding window of recent blocks to account for potential forks, as done by Dennis et al. (2016). Following such a design, robots could effectively operate using this *pruned version* of the blockchain, reducing their storage requirements. When a connection to external infrastructure or more capable nodes is available, the blockchain history or snapshots can be stored and retrieved on demand.

Light clients Ethereum light clients, or Bitcoin's simple payment verifiers (as examples), do not store full blocks. Instead, they only validate and maintain block headers in runtime memory[2] and are able to verify inclusion of transactions via Merkle proofs provided by full nodes (approximately 500 bytes each). This allows robots with limited storage to participate alongside more capable peers that host the full blockchain: they can query full nodes for specific transactions and validate responses without needing to trust them.

Commitments Algorithms should minimize what is stored on-chain. In general, blockchains are not designed to store bulk raw data (Parisi et al. 2025) as this increases storage and bandwidth requirements across all nodes. Instead, blocks can host *commitments* (e.g., cryptographic hashes) that act as stable identifiers for data kept locally by each robot, as done by Nishida et al. (2018). This preserves the benefits of a shared, tamper-evident state while keeping data private and local. Consistency checks can then be performed off-chain by exchanging the data when needed and validating it against the on-chain commitments. In Chapter 7 we discuss this approach further, given the data-heavy nature of the federated learning and Swarm-SLAM applications.

[2] 73 MB in Bitcoin in comparison to 690 GB for the full blockchain, as of November 2025

(B) Communication growth

Blockchains generally impose light communication requirements on nodes to remain scalable. Their consensus protocols tolerate substantial participant unavailability, and all information exchanges (blocks and transactions) occur via peer-to-peer gossip—without requiring receipt acknowledgments or routing to any particular node. This design keeps costs linear with the number of nodes. To reduce inefficiencies caused by conflicting blocks, blockchains tend to use predictable block periods that provide sufficient time for disseminating the blocks across the network[3].

However, swarm networks are highly dynamic. Communication is often local and intermittent, and robots' faults and mobility can create prolonged periods of disconnection. These disconnections can disrupt block production because typical consensus protocols were not designed for such conditions. For example, in a partitioned swarm, proof-of-work-based consensus may become vulnerable to local 51 % attacks within a subnetwork, while proof-of-authority can stall block production when authorized validators are unreachable. Therefore, it is necessary to make adequate design choices regarding parameters, protocols, and robot behaviors.

Parameter adjustment Blockchain consensus protocols can be tuned to specific applications. For example, in proof-of-authority, one can adjust parameters such as the relative difficulty (or weight) assigned to blocks created by the designated round leader versus non-leaders, the minimum delay before a non-leader may propose an out-of-turn block, and the *block period*—the minimum time between consecutive blocks (see Chapter 4 for a detailed description of the proof-of-authority protocol). Among these, the *block period* B_p is particularly important, as it directly influences the system's responsiveness, scalability, and communication overhead. In applications where the blockchain is used for processes with slow dynamics (e.g., consensus, event logging, or reputation management), sparse block generation can help scaling to larger swarms.

Accordingly, in Chapter 5, a consensus task is presented where $B_p = 15\,\mathrm{s}$, while in Chapter 7 – Section 7.2, it is reduced to 2 s to enable real-time coordination and enhance the responsiveness of decision-making within a robot swarm in a foraging task. However, it is observed that reducing the block period increases communication costs, as blocks are generated more frequently and synchronizing the blockchain becomes more demanding. Therefore, a trade-off exists between costs and latency, indicating that blockchain applications in multi-robot systems should be reserved for high-level decision-making and security-critical applications rather than for lower-level control of individual robots—unless improvements are made to the protocol itself.

[3]Approximately 10 minutes in Bitcoin and 12 seconds in Ethereum, as of November 2025

Protocol development Recent research has focused on developing protocols that are tailored to swarm robotics. One prominent approach, alternative to blockchains, is based on directed acyclic graphs (DAGs). Tran et al. (2019) introduced SwarmDAG, a partition-tolerant protocol, which organizes transactions in a tree-like structure rather than a linear chain. In this approach, when a swarm becomes divided into disconnected sub-networks, robots can first reach a local consensus and later achieve global consensus once the networks are reconnected, ensuring eventual data consistency. Similarly, Keramat et al. (2023) and Salimpour et al. (2023) address the partitioning problem by leveraging the existing DAG-based distributed ledger IOTA.[4]

While DAGs offer potential for swarms with intermittent connectivity, their development remains preliminary, lacking tests on physical robots. The advantage of enhancing responsiveness by appending transactions asynchronously also complicates the achievement of a consistent global state, since DAGs lack the straightforward ordering and immutability of linear blockchains. To address this, IOTA has relied on coordinator nodes, sacrificing full decentralization. The state-of-the-art Narwhal&Tusk (Danezis et al. 2022) protocol combines blockchain BFT consensus (Narwhal) with a DAG-based mempool (Tusk) to improve availability and reduce latency when networks are partitioned, while ensuring eventual global convergence. This design shows promise but remains unexplored in robotics contexts.

Improving connectivity Communication challenges can also be addressed from a networking perspective, for instance, by improving connectivity and robustness through redundancy (k-robustness, Zhang et al. 2015) or through group behaviors, where robots actively improve connectivity through coordinated motion strategies such as flocking (Saulnier et al. 2017), formation control (Dimarogonas and Johansson 2008), or the use of messenger robots that relay information across disconnected parts of the swarm (Zhao et al. 2004). A recent line of work employs control barrier functions to constrain robot motion and prevent network fragmentation (Capelli et al. 2021), even in adversarial environments (Cavorsi et al. 2024).

(C) Computation growth

Although blockchains are not compute-intensive by default, their computational footprint depends heavily on the applications. Specifically, it depends on the algorithms and logic encoded in their internal state and that of smart contracts. The design of contracts can significantly affect the overall costs and scalability of the system. While smart contracts can potentially encode top-down or global (swarm level) decision processes (see Section 3.3 Opportunities), one must avoid deploying

[4]See `https://www.iota.org/`, accessed November 2025

algorithms that are computationally expensive or scale poorly. For example, some multi-robot task allocation algorithms display $\mathcal{O}(n^3)$ complexity (Chopra et al. 2017), or even become NP-hard in realistic settings (Gerkey and Matarić 2004), making them unsuitable for swarm applications.

Blockchains are better suited for lightweight coordination schemes and simple rules, or market or auction-based task allocation approaches (Dias et al. 2006, Lagoudakis et al. 2004), where the blockchain could act as a decentralized marketplace, aggregating bids from robots and managing economic incentives to promote efficient and cooperative behavior within the swarm. In Chapter 7 – Sections 7.2 and 7.5 we present approaches to coordinate swarm foraging through both simple blockchain-enforced rules and economic interactions among robots. These approaches improve coordination and incentivize cooperation by leveraging blockchain features.

In summary, while the blockchain by itself imposes low computational requirements, the overall system demands depend critically on the complexity of the smart contracts and on the distribution of computation across the swarm, which in turn depend on the task itself. Practical designs should delegate heavy computation to off-chain local processes among robots, while reserving the blockchain for lightweight coordination, verification, and incentive alignment.

3.2.3 Oracle problem

To operate and scale globally, blockchain systems are decentralized, transparent, and trustless, relying on large networks of nodes to independently verify that new blocks follow consensus rules and that user transactions are valid. A fundamental limitation follows from this design: *blockchains cannot natively incorporate information unless it is algorithmically verifiable by every node.* Antonopoulos and Wood (2018) discuss how this limitation extends to smart contracts that require accurate real-world data for decision-making, such as exchange rates for decentralized finance, seismic measurements for disaster insurance, or environmental indicators for managing smart cities and natural resources (see Box 3.1).

In blockchain systems, the bridge between off-chain facts and on-chain logic is provided by *oracles*, a term metaphorically inspired by the oracles of Greek mythology—entities regarded as mediators of divine wisdom between the gods and humans.[5]

In the example above, a prospective smart contract could obtain the required information from a trusted environmental agency or autonomous software that

[5]The original introduction of this metaphor in computer science literature is Alan Turing's concept of "oracle machines," hypothetical devices capable of solving problems beyond the limits of algorithmic computation (Turing 1939).

Box 3.1: Example of the Oracle problem

Consider a smart contract that distributes rewards to local landowners for maintaining a sustainable forest (e.g., using funds originating from the sale of carbon credits). In principle, such a contract could implement transparent, immutable rules that guarantee landowners direct and equitable access to funds, independent of shifting politics, opaque bureaucracies, and financial intermediaries. However, to determine rewards, the contract requires reliable information about the state of the forest: *Are biodiversity levels increasing? At what rate are trees being felled? Has a fire occurred?*

processes satellite images—both of which acting as oracles. However, relying on oracles to obtain accurate real-world data reintroduces elements of trust and centralization, potentially undermining blockchain's core properties such as automation, censorship resistance, and transparency. This is known as **blockchains' oracle problem** (Caldarelli 2020).

Implementing reliable oracles remains a *critical open problem* affecting not only public blockchains, but also our robotics applications, since sensing and acting upon real-world states (e.g., environmental conditions or successful task completion) is at the core of many swarm tasks. To avoid introducing single points of failure and trust through centralized oracles, decentralized oracle networks (DONs) were introduced.

Decentralized Oracle Networks (DONs) reduce the need for single-party trust by aggregating reports from multiple contributors to reach collective agreements on off-chain facts. The reliability of these agreements hinges on two pillars: cryptographic assurance (e.g., authenticated sensing, secure aggregation, attestations) and incentive alignment, which rewards truthful reports and penalizes manipulation attempts. Breidenbach et al. (2021) proposed Chainlink, a network consisting of automated software oracles that retrieve and process online information such as asset prices and news events. A DON can also consist of multiple hardware oracles that collect data directly from the physical environment using sensors. A prominent example is Helium (Haleem et al. 2024), whose IoT network provides wireless coverage while gathering geolocation and connectivity data for on-chain applications. Beniiche (2020) proposes a taxonomy of existing oracle solutions based on their underlying trust models, ranging from authoritative to decentralized consensus-based approaches, and on the nature of their participants (e.g., software, hardware, and even human). Chapter 2 provides a review of existing DONs. However, despite growing demand for transparent and trustworthy data, existing decentralized oracles, hardware ones in particular, lack the autonomy, flexibility, and scalability required for most applications.

Swarm Oracle In Chapter 6, we present a decentralized oracle network, that we call Swarm Oracle, composed of autonomous robots that use onboard sensors and peer-to-peer communication to collectively verify real-world data and deliver it to blockchains—both to public blockchains and to robot-hosted blockchains (Pacheco et al. 2025†). Swarm Oracle thus addresses the oracle problem of blockchains, introducing trustworthy information to the blockchain that the swarm can exploit for self-organization. Beyond self-organization, it is also a novel application of swarm robotics that leverages their built-in decentralization, mobility, and fault tolerance to flexibly respond to information requests from public blockchains, even in remote locations. For instance, robots could monitor the forest mentioned in the example above to provide trustworthy information for decision-making in public smart contracts, as shown in Figure 3.1.

Unlike typical cooperative robot swarms, Swarm Oracle integrates robots from multiple stakeholders to protect the system from biases introduced during the manufacturing or programming of the robots. However, this introduces potential adversarial and heterogeneous behavior among the robots. To address this, Swarm Oracle employs a BFT protocol and reputation system, where untrusting robots can operate together to reach collective agreements of higher quality than individual estimates, despite attacks orchestrated by large proportions of the robots. The reputation system allows Swarm Oracle to self-heal from faults and attacks, enabling autonomous operations.

3.3 Opportunities

Blockchain technologies open several avenues for advancing swarm robotics in both research and applied settings. In Dorigo et al. (2024†) we identified five potential opportunities provided by the integration of blockchain and mobile multi-robot systems: (i) enabling self-governance, (ii) facilitating economic interactions, (iii) establishing security, trust, accountability and compliance, (iv) ensuring data consistency, and (v) supporting top-down system design.

As discussed in the context of the scalability challenge, tasks that require fast and responsive decision making are typically best handled off-chain. Consequently, the opportunities identified here do not concern the low-level control of individual robots; instead, blockchain and smart contracts are presented as *complementary mechanisms* to traditional control approaches that enhance the swarm's self-organization and coordination capabilities.

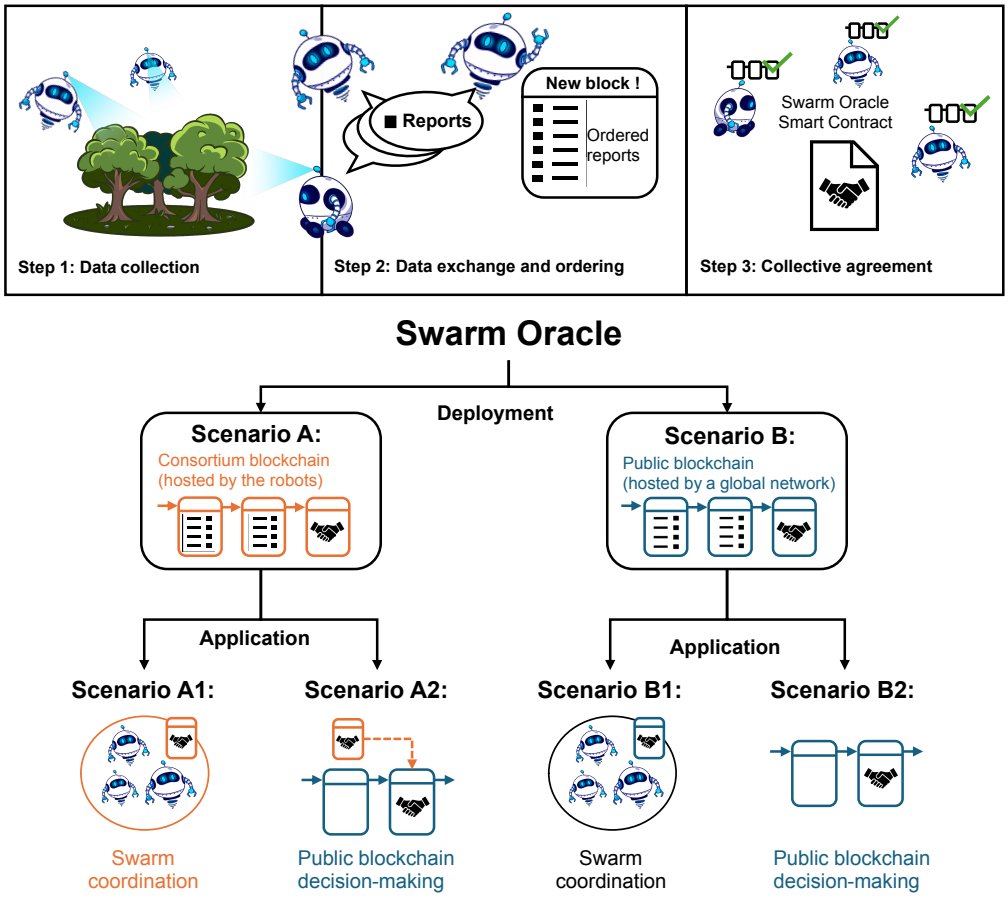


Figure 3.1: Swarm Oracle. ***Execution steps.*** **(1)** robots observe the environment to gather data, **(2)** send their observation reports to a blockchain that orders the reports, and **(3)** reach a collective agreement using a protocol implemented as a smart contract. ***Deployment.*** The smart contract can be deployed either through: **(A)** a blockchain that is hosted by the robots, or **(B)** a public/external blockchain. ***Application.*** The secure agreements enabled by the smart contract can be used for: **(A1/B1)** internal decision-making (by the swarm itself), or **(A2/B2)** by smart contracts or decision processes based on public blockchains. With respect to the Architectures presented in Chapter 2, scenario A follows *Architecture 1*, scenario B follows *Architecture 2*, and scenario A2 is an example of *Architecture 3*, where the smart contract is deployed on a robot-hosted sidechain, and only the collective agreements are uploaded to the external blockchain.

3.3.1 Self-governance

Blockchain technology offers a timely opportunity to establish self-governing robot swarms. Beyond coordination at the task level, blockchains and smart contracts can provide the means for swarms to organize their internal structure—particularly, defining roles and hierarchies, managing membership (admission, departure, and exclusion), and recording the capabilities and reputations of heterogeneous robots across tasks. Self-governance may also extend to the stewardship of shared resources: when many robots operate in a common environment, they must coordinate access to scarce assets such as physical space, wireless bandwidth, computational resources, charging stations, or consumable materials.

An interesting and novel swarm-governance paradigm is that of *open-swarms*: collectives of robots from multiple stakeholders that must coordinate tasks, roles, and access to shared resources under potentially competing objectives. Closely related governance and resource stewardship are studied in blockchain research through *Decentralized Autonomous Organizations* (DAOs, Lustenberger et al. 2025).

(A) Open-swarms

Self-governance is particularly relevant for ***open-swarms*** (Pacheco et al. 2025†, Reina 2020, Strobel et al. 2020), which are composed of robots that belong to different stakeholders and were developed by different companies. They are "open" in that robots may join or leave the swarm dynamically and act on behalf of stakeholders (individuals, companies, or governmental bodies), bringing heterogeneous objectives and potential competition. Open-swarms introduce novel coordination challenges rarely studied in swarm robotics literature. These challenges include incentive alignment, fair allocation of shared resources, truthful (and auditable) information sharing, and resilience to Byzantines or free-riders—robots that exploit the work of others.

(B) Decentralized Autonomous Organizations

An emerging theme in blockchain research is the development of ***Decentralized Autonomous Organizations*** (DAOs, Box 3.2), governed by transparent rules encoded in smart contracts that define internal policies and regulate member interactions (El Faqir et al. 2020, Hassan and De Filippi 2021, Wang et al. 2019). While DAOs are typically composed of human participants, they can also be conceived as collectives of computational agents, robots, or hybrid assemblies of humans and artificial entities (Cardenas et al. 2021). In the context of swarm robotics, DAOs could be developed to regulate access control by specifying the conditions for robot admission and removal, define ownership and accountability

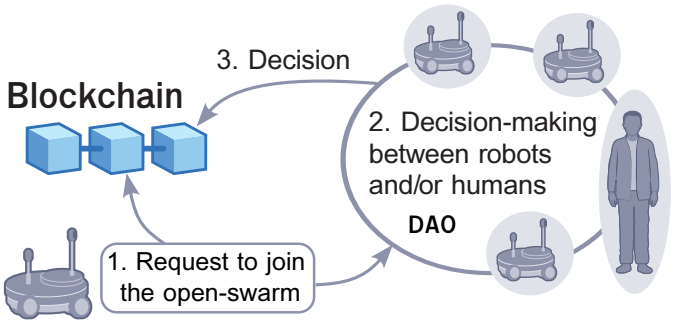


Figure 3.2: Governance of an open-swarm through a blockchain-based DAO. Image adapted from Dorigo et al. (2024†).

structures, and coordinate participants toward shared objectives through incentive and governance mechanisms, as represented in Figure 3.2.

Box 3.2: Decentralized Autonomous Organization (DAO)

A DAO is a network that enables participants to coordinate and self-govern through a set of self-executing rules and mechanisms deployed as smart contracts on a public blockchain (Hassan and De Filippi 2021). The governance of a DAO is decentralized, meaning that no single entity holds central control over its operations. Instead, decision-making and resource management are carried out collectively by members according to transparent, algorithmically enforced rules.

Key characteristics:

- **On-chain governance:** Organizational policies and interactions among members are defined in code and executed automatically via smart contracts.
- **Transparency and auditability:** All operations and decisions are recorded on a public blockchain ledger, ensuring verifiability.
- **Token-based participation:** Members typically hold governance tokens that represent voting rights or stakes in the DAO's resources.
- **Autonomy:** Once deployed, a DAO operates without oversight, subject only to its coded rules and the collective decisions of its members.
- **Decentralization:** No central authority manages the DAO; control is distributed across its participants and is backed by the public blockchain.

While early notions of DAO focused on autonomous economic operation, modern DAOs generalize this concept to encompass diverse forms of digital and physical collaboration and resource sharing—not only among humans, but also among artificial agents such as robots and AI.

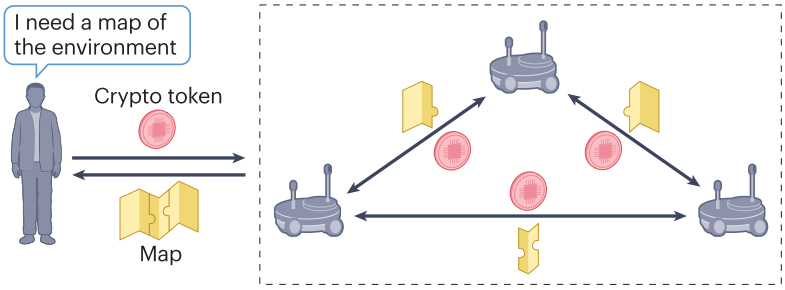


Figure 3.3: Economic interactions among humans and robots through blockchain crypto tokens. Image adapted from Dorigo et al. (2024†).

3.3.2 Economic interactions

The original application of blockchains as cryptocurrency ledgers introduced monetary transfers which are fully automated and digital, without relying on financial intermediaries. This enables scalable peer-to-peer transactions and micro-payments globally (Antonopoulos et al. 2021, Nakamoto 2008, Poon and Dryja 2016).

By supporting automation and micro-payments, cryptocurrencies have the potential for widespread adoption in machine economies, where robots and other artificial systems participate as economic agents alongside humans (Castelló Ferrer et al. 2023). As shown in Figure 3.3, humans could, for instance, hire mobile multi-robot systems to perform agricultural, environmental, or industrial tasks and compensate them directly via crypto payments, establishing *robots-as-a-service* business models (Castelló Ferrer et al. 2021a). Conversely, robots could autonomously pay humans, external software or other robots for services such as maintenance, energy recharges, or cloud storage.

In the following, we examine how economic principles implemented through blockchain-based systems can form the foundation for new approaches to swarm design and organization. We then introduce *Decentralized Physical Infrastructure Networks* (DePINs), a novel class of blockchain-based infrastructure that extends digital economies by integrating physical devices that provide services.

(A) Economics-aligned robot behaviors and swarm design

The coordination of large populations of autonomous agents under limited information is a fundamental challenge shared by both economics and swarm robotics. Seminal work in economic theory by Hayek (1945) shows that scalable and adaptive coordination can emerge without centralized control by delegating control to agents that are familiar with local conditions. Information travels through the system through price signals that aggregate dispersed knowledge and guide individual behavior, enabling globally coherent outcomes despite uncertainty and

heterogeneity. *Price* is a scalar quantity that encodes information about the production, availability, scarcity, and demand for resources, allowing large-scale economic systems to coordinate efficiently without centralized oversight or agent's global awareness—both of which would otherwise be computationally and organizationally intractable.

Market-based approaches have been explored to achieve distributed and scalable coordination of multi-robot systems (Dias and Stentz 2000, Dias et al. 2006). In particular, Zlot et al. (2002) propose a market-based coordination framework for multi-robot exploration that significantly improves coverage efficiency and robustness compared to greedy and non-communicating approaches; Gerkey and Mataric (2002) and Lagoudakis et al. (2004) propose auctions where robots bid on tasks on the basis of their individual cost estimates; and Zlot and Stentz (2006) use peer-to-peer trading to decompose complex tasks into smaller tasks, promoting robot specialization through market incentives. However, adoption in swarm robotics has been limited, owing to (i) the difficulty of implementing a market or auctions without a centralized coordinator and (ii) a mismatch between individualist/game-theoretic mechanisms and cooperative/goal-aligned designs inspired by eusocial insects.

Within a robot swarm, a blockchain enables the formation of an internal economic system without centralized economic controls. It can therefore act as the foundation to apply novel system design philosophies from mechanism design, game theory, and auction systems to yield emergent self-organization, in which robots autonomously allocate tasks based on skills, costs, and expected rewards—potentially rewarding indirect contributions (e.g., robots that assist peers in achieving their tasks, Givigi Jr. and Schwartz 2006.

Economics-inspired design gains relevance in *open-swarms*, as it can promote trust and cooperation among robots belonging to different stakeholders. Robots may trade information or services in exchange for crypto tokens: For example, aerial robots could sell environmental data or mapping information to ground robots, while ground robots could offer transport or energy services in return. Chapter 7 – Section 7.5 explores an information marketplace that promotes social navigation when robots are untrusting and unreliable. We explore how the economic incentives embedded within smart contracts play a critical role in promoting cooperative behaviors and penalizing greed or malicious actions (Van Calck et al. 2023[†]).

(B) Decentralized Physical Infrastructure Networks

Decentralized Physical Infrastructure Networks (DePINs) (Box 3.3) extend blockchain coordination and resource management into the physical realm. In DePINs, autonomous devices managed by individuals collectively build and maintain shared infrastructure, such as wireless networks (Haleem et al. 2024), electricity

Box 3.3: Decentralized Physical Infrastructure Network (DePIN)

DePINs create decentralized service economies by harnessing the collective capabilities of simple devices maintained by many individuals. In this model, devices provide revenue while delivering useful services to a community. This novel opportunity extends blockchain utility beyond digital ledgers, enabling decentralized management of physical infrastructure.

Key characteristics:

- **Collective ownership:** Token incentives are provided to deploy and maintain devices, creating self-organized networks with distributed ownership.
- **Low operating costs:** Self-governance, shared infrastructure, and incentive alignment reduce upkeep costs, barriers to entry, and corruption.
- **Privacy, security, and resilience:** Decentralized architectures avoid single points of failure and sustained services despite loss of devices; blockchain integration enforces data integrity and access control.
- **Openness and innovation:** Open participation fosters competition, accelerates innovation, and promotes equitable access to services.

Several projects tackle connectivity, electricity, cloud computing, environmental sensing, and physical services. *Robots are particularly well-suited to participate in DePINs:* their autonomy, mobility, and sensing capabilities allow them to deliver localized, on-demand services efficiently.

grids (Pop et al. 2018), edge computing (e.g., for AI and data-intensive applications, Yang et al. 2019, and distributed sensing (for environmental, industrial, or urban monitoring[6]). Despite being a recent concept, with the first peer-reviewed article mentioning DePIN appearing in 2025 (Lin et al. 2025), several startups are piloting DePIN innovation, providing opportunity for the integration of robotics systems. Unlike traditional service providers (e.g., ISPs, carriers, or privately owned industrial fleets), DePINs are built on core blockchain principles: transparency, decentralization, and minimized trust through peer-to-peer interactions among clients, devices, and stakeholders.

This provides an opportunity for robotic systems. Robots have the potential to play a central role in DePINs: their mobility and autonomy support on-demand services that are localized and economically efficient in serving communities such as those in remote locations where infrastructure is sparse. Swarms in particular, adhere to DePINs principles by being composed of inexpensive and low-cost devices, that when networked, can provide services with high reliability and flexibility.

[6]For example, collecting noise data (Silencio Network 2025) and GNSS data (Horton et al. 2023)

3.3.3 Security, trust, accountability and compliance

Swarms have traditionally been designed under a cooperative paradigm in which robots—although unreliable and prone to faults—do not act in self-interest nor intentionally attempt to disrupt collective goals. However, this assumption may not hold in real-world deployments, where (i) hardware faults are unpredictable and erratic, (ii) robots may be compromised by adversaries, and (iii) swarms may include robots from stakeholders with conflicting interests.

Research in swarm robotics addressing such conditions remains limited. Several algorithms that model non-ideal behaviors[7] break down when confronted with Byzantine robots that deliberately exploit system vulnerabilities. Strobel et al. (2018) demonstrated that, under certain circumstances (e.g., a Sybil attack), even a single Byzantine robot can compromise the entire swarm, effectively acting as a single point of failure in an otherwise decentralized and fault-tolerant system.

This lack of built-in security has hindered the deployment of swarm robotics in real-world applications, where systems must operate safely, comply with applicable laws, and avoid causing harm to humans, the environment, or other robots (Hunt and Hauert 2020, Winfield and Nembrini 2006).

Blockchain protocols provide a foundation for achieving security, accountability, and compliance in swarm robotics (Castelló Ferrer 2023). Through digital signatures and consensus mechanisms resistant to double counting and Sybil attacks, blockchains enable the recording and verification of inter-robot communications, allowing inconsistencies in robot behavior to be detected and malicious agents to be isolated or sanctioned (Figure 3.4). More broadly, the immutability and transparency of blockchain logs supports the development of responsible AI and robotics (Müller 2025), strengthening public trust through verifiable data trails that facilitate compliance auditing, liability assignment, and mechanisms for autonomous fault recovery.

(A) Fault detection

A typical strategy to ensure compliant behavior is to detect faults or inconsistencies in the robots' behavior (Tarapore et al. 2019), either through self-checks (endogenous detection) or collective behaviors to detect faults (see Chapter 2). However, self-checks may fail when faults are erratic or intermittent, and collective methods for exogenous detection (Tarapore et al. 2017) are vulnerable to agents that emit false or conflicting reports. In other words, traditional methods fail when robots exhibit *Byzantine behaviors*.

[7]For example, models of zealot, stubborn, contrarian, or noisy robots—see Chapter 2 and Khaluf and Hamann 2019, Masi et al. 2021, Prasetyo et al. 2018, Valentini et al. 2016.

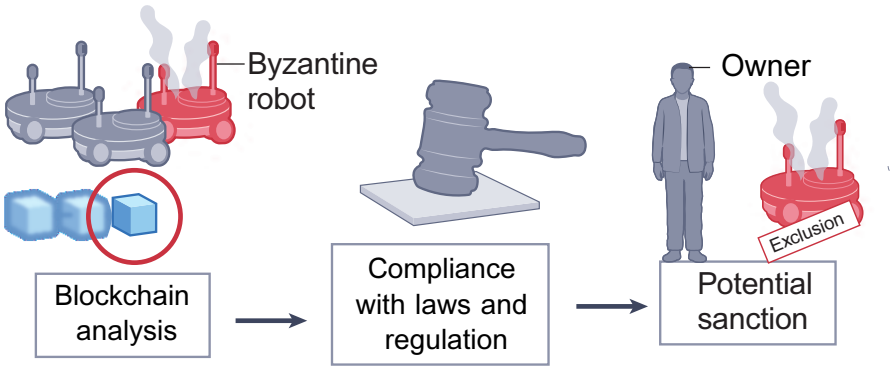


Figure 3.4: Achieving compliant swarms through blockchain analysis. Image adapted from Dorigo et al. (2024†).

Blockchain facilitates the detection of inconsistencies or deceitful reporting among robots, since messages are cryptographically signed and timestamped. In Chapter 5, we deploy a smart contract that aggregates sensor readings from the robots and implements outlier detection mechanisms to identify Byzantine robots that send extreme values—which can then be excluded from the swarm. While this approach enhances the swarm's robustness against simple Byzantine behaviors, fault-detection does not work universally and may lead to false positives (Lynch 1996). Indeed, a smart attacker could reverse-engineer the defense mechanisms, or coordinate several robots to mislead the fault detectors. For this reason, Chapter 6 implements a Byzantine fault-tolerant (BFT) algorithm that does not rely on fault detection.

(B) Byzantine fault tolerance

BFT algorithms are designed to withstand arbitrary, potentially malicious behavior up to a given number of Byzantine participants. Rather than tailoring defenses to specific failure modes, BFT systems aim to remain correct under *any* action incurred by Byzantine agents in a distributed or decentralized network. The concept originates from fault-tolerance in computer science literature (Lamport et al. 1982), and has been also applied extensively in robotics (Bouzid et al. 2009a, Défago et al. 2006, Deng et al. 2021, Peleg 2005, Strobel et al. 2018).

Unlike existing BFT approaches in robotics reviewed in the Chapter 2,[8] blockchains provide fundamentally different and more general mechanisms for achieving consensus and securing robot interactions, without relying on assump-

[8]Namely, fault-tolerant rendezvous algorithms that do not require explicit fault detection (Park and Hutchinson 2017); and resilient consensus methods that rely on robust networks (LeBlanc et al. 2013)

tions about network structure.

In a blockchain network, messages propagate on a best-effort basis among peers, and nodes may leave or rejoin without halting overall progress (Nakamoto 2008). When rejoining, a node simply adopts the strongest chain. This process yields a form of BFT consensus that is probabilistic: confidence in the finality of a block increases as additional blocks are appended. This contrasts with deterministic BFT algorithms, which provide immediate finality but incur higher communication overhead.

Building on these properties of blockchain-based consensus, in Chapter 6, we deploy a BFT consensus protocol within a robot swarm that guarantees *safety and liveness*: as long as no more than one-third of the robots are Byzantine, the swarm will neither reach incorrect nor diverging agreements (safety), and the swarm will eventually come to an agreement (liveness). This type of global-level security assurances is rarely seen in swarm robotics research.

(C) Reputation systems and resilience

Whereas fault detection focuses on preventing failures, another strategy is to mitigate their impact over time, particularly when robots exhibit recurring faulty behavior. One solution is to develop *reputation systems* that quantify each robot's trustworthiness within the swarm. Although this may not help towards preventing catastrophic or instantaneous failures, it may enhance resilience over time, providing an adaptive and autonomous response to faults without requiring external supervision (Pacheco et al. 2025†). In Chapter 6, we demonstrate this effect: our reputation system mitigates the influence of Byzantine robots, allowing the swarm to recover and maintain secure and efficient operations.

Blockchain smart contracts play a crucial role in this process, as they store tamper-proof reputation values in the form of crypto tokens and define the rules for token (re)distribution among robots. A key feature of reputation systems is their minimal structure, consisting primarily of a mapping between identities and reputation values, which can be flexibly adapted to different applications while maintaining manageable storage demands on the robots.

Although researchers have begun exploring how Byzantine robots can be neutralized through blockchain-based reputation systems in specific tasks (Castelló Ferrer et al. 2021b, Strobel et al. 2018, 2020, 2023†, Van Calck et al. 2023†, Zhao et al. 2023†), the general principles for programming smart contracts to achieve this remain an open question. In Chapter 6, robots that send reports accepted by Swarm Oracle gain reputation tokens, while inactive or rejected robots lose tokens. Robots' reputation tokens are then used by the smart contract as weights when aggregating information—reducing the impact of malicious or unreliable robots. Another approach is to leverage the fact that reputations are public to

promote individual decision-making, allowing robots to selectively collaborate with, or disregard information from, peers based on reputation—as illustrated by the Information Marketplace in Chapter 7.

(D) Trustless swarm design

Designing robot swarms that operate without relying on mutual trust among robots represents a promising new paradigm made possible by blockchain technology. This paradigm aligns with the concepts of the Internet of Robotic Things (Simoens et al. 2018) and open swarms by extending swarm coordination to real-world environments that involve heterogeneous robots and multiple stakeholders. However, it also introduces novel challenges that remain largely unexplored in the literature, including Sybil resistance in open systems, identity bootstrapping without central authorities, handling dynamic swarm sizes, designing for self-interested and heterogeneous agents, and balancing accountability with privacy.

Blockchain offers a general-purpose coordination layer to help address some of these challenges. Through smart contracts, swarms can execute shared code and ensure secure peer-to-peer interactions such as monetary transfers, escrowed guarantees, and reputation mechanisms. These can facilitate self-organization and decision-making among mutually untrusting agents. Chapter 6 presents Swarm Oracle as the first swarm system explicitly designed under this paradigm, enabling the deployment of robot collectives as trustworthy oracle systems that provide reliable data to blockchains. In a Swarm Oracle, distributing robots across different stakeholders is essential to ensure that no single entity can compromise the integrity of the data by controlling the entire swarm.

(E) Blackbox logging

Unlike the previous topics, which focused on achieving secure, accountable, and compliant swarms through internal mechanisms, blockchains can also enable *external analysis and oversight* during and after deployment. Online monitoring of traditional swarms is costly due to system complexity and large robot counts, and post-mission logs gathered from a single robot or subset (e.g., only those that returned from hazardous areas) may be incomplete or corrupted. In contrast, the blockchain's immutability and swarm-wide replication enable external observers to (i) monitor blocks forwarded by any one robot rather than listening to all inter-robot traffic, and (ii) retrieve the entire blockchain from a single robot after an experiment, with strong guarantees against tampering or data loss. These records support behavior validation during execution and provide an auditable chain of actions for legal review when noncompliance with applicable laws is alleged (Danilov et al. 2018).

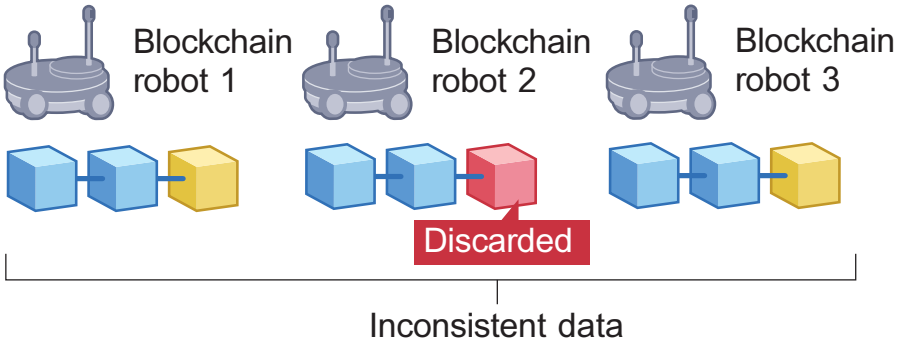


Figure 3.5: Achieving data consistency through blockchain-based aggregation. Image adapted from Dorigo et al. (2024†).

3.3.4 Data consistency

Many multi-robot tasks rely on maintaining a consistent state of shared data; otherwise, robots may compute conflicting action plans, leading to inefficiencies or even mission failure. Achieving data consistency is challenging due to potential component failures, communication delays, and adversarial behavior, and becomes even more difficult in decentralized swarm robotics, where global synchronization mechanisms are typically absent. Two recurring issues are (i) *double counting*, where the same message is processed multiple times, and (ii) *message ordering*, which is essential for consistent data aggregation.

As discussed previously, blockchains can address both issues while simultaneously protecting against malicious attacks. Using the general methods presented throughout this thesis, robots can submit data to smart contracts via transactions, ensuring that all data is aggregated and replicated consistently across the entire swarm. When data packets are too large to store directly on-chain, robots can instead maintain local copies and commit cryptographic hashes to the blockchain—thereby proving data existence at a specific point in time and identifying the robot responsible for storing it.

To ground these ideas, we examine two data-intensive tasks: *collective mapping* and *federated learning*. In conventional multi-robot systems, these tasks are often executed in centralized or insecure ways. In Chapter 7 – Sections 7.3 and 7.4, we present two deployments of robot swarms that use the blockchain to decentralize and secure data aggregation, improving upon previous non-blockchain approaches such as Swarm-SLAM (Lajoie and Beltrame 2024) and Flow-FL (Majcherczyk et al. 2021).

(A) Collaborative mapping

While the extensive literature on single-robot simultaneous localization and mapping (SLAM) has focused primarily on robustness to noise and bias in sensor data (Aulinas et al. 2008, Kümmerle et al. 2009), recent work has increasingly shifted toward collaborative SLAM, in which multiple robots cooperate to build a shared global map (Saeedi et al. 2016a). Multi-robot SLAM offers significant advantages in terms of efficiency, robustness, and parallelization; however, it also introduces challenges related to scalability and data consistency.

A central challenge is the aggregation of the map—typically represented as robot trajectories and pose graphs (Kim et al. 2010). This process requires each robot to share environment perception data, which must then be merged into a global map through data fusion or optimization. Robots also contribute loop closures, which are constraints linking their trajectories when they observe the same region of the environment. Most existing methods for generating a global map rely on centralized architectures, and only recently have researchers begun exploring decentralized or swarm-based deployments (Kegeleirs et al. 2021).

In swarm-SLAM (Lajoie and Beltrame 2024), robots exchange data and loop closures locally, and a designated robot periodically performs the optimization. However, the approach provides no guarantees of map convergence, lacking both a consensus mechanism and protection against Byzantine robots. In Moroncelli et al. (2024†) we demonstrate that incorrect loop closures contributed by Byzantine robots introduce invalid constraints that severely degrade the optimized global map. We present the blockchain-based validation scheme shown in Figure 3.6, and test it in a simulated SLAM scenario. These results are shown and discussed in Chapter 7 – Section 7.4.

(B) Federated learning

Federated learning (FL), originally proposed by McMahan et al. (2017), is a distributed machine learning paradigm in which multiple devices collaboratively train a shared model without exchanging raw data. Each device independently trains a local model and transmits only its learned parameters, which are then aggregated to update the global model, as illustrated in Figure 3.7. This approach preserves data privacy, reduces communication overhead, and enables efficient parallelization of training across distributed agents (Wen et al. 2023). These properties make FL especially attractive for mobile robotics (Xianjia et al. 2021), where robots gather heterogeneous sensor data while operating under strict bandwidth and communication constraints. However, most FL schemes rely on a centralized server to aggregate parameters, which limits their applicability to swarm systems.

In Chapter 7 – Section 7.3, we demonstrate how a blockchain can decentralize

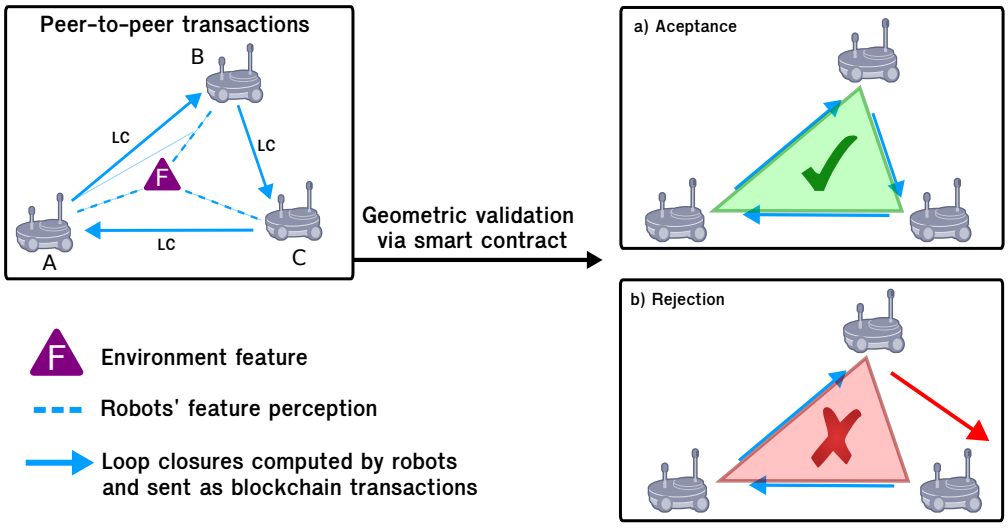


Figure 3.6: Data consistency: Swarm-SLAM. The robots A, B, and C send blockchain transactions containing loop closures (LCs) between their trajectories (blue arrows), which are generated when robots identify the same purple feature F in the environment. The smart contract validates each trio of LCs: If the three LCs generate a valid triangle, they are approved and the robots receive a reward; otherwise they are dropped

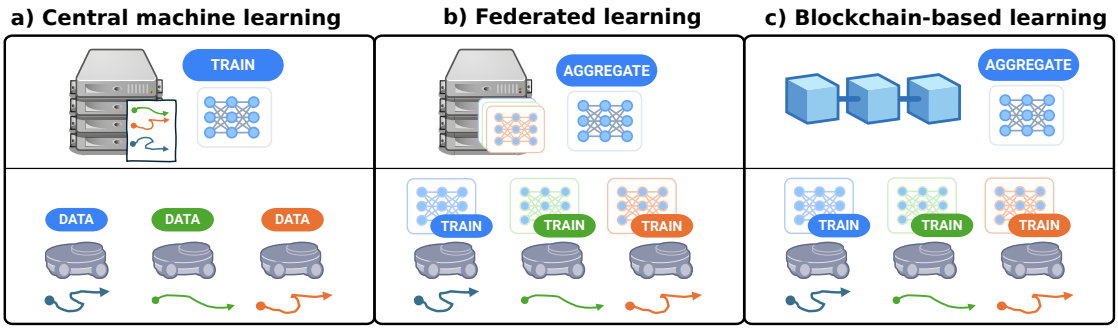


Figure 3.7: Data consistency: Federated learning. Three models are possible:**(a)** In centralized machine learning, robots transmit raw data to a central server which trains the model. **(b)** In federated learning, only the locally trained neural network models are sent to the server, while data remains privately stored on each robot. **(c)** Using a blockchain, the models can be aggregated securely without relying on a server. In the figure, robots collect trajectories by observing themselves and their neighbors to train local models, which are then aggregated using a blockchain smart contract (Pacheco et al. 2024a†).

and secure FL within a simulated swarm (Pacheco et al. 2024a†). Each robot collects motion data from nearby agents and trains a local motion model consisting of 2,000 parameters. The resulting models are packaged as blockchain transactions and aggregated on-chain using a smart contract and the standard `FedAvg` algorithm (McMahan et al. 2017), enhanced with outlier rejection mechanisms and contribution weights. Building on previous research by Majcherczyk et al. (2021), we show that our system maintains robustness since models with the lowest performance are discarded prior to aggregation, enabling the global model to converge even in the presence of Byzantine or faulty robots.

3.3.5 Top-down swarm design

Robot-swarm controllers are traditionally designed *bottom-up*: developers iteratively test and refine individual behaviors and local interaction rules until the desired collective dynamics emerge (Brambilla et al. 2015). This design paradigm is rooted in our understanding of how natural collectives of insects, fish, or birds self-organize at remarkable scales.

More recently, however, researchers have explored how structured organization plays a role in the design of robot swarms, including explicit hierarchical roles (Varadharajan et al. 2024) and emerging topologies (Zhu et al. 2024). One influential line of work investigates *mergeable nervous systems* (Mathews et al. 2017), where swarms can dynamically split or merge into subgroups that employ localized hierarchical or centralized decision-making.

Blockchain technology and smart contracts extend these ideas by enabling a new form of hierarchical yet decentralized control, in which smart contracts coordinate group-level planning to shape individual action policies. Ideally, this approach provides high-level oversight while retaining robots' autonomy, allowing them to react to disturbances and exploit local information without undermining the swarm's robustness, adaptability, and responsiveness (Pacheco et al. 2022†), as illustrated in Figure 3.8. By leveraging reputation systems, on-chain behavior analysis, and incentive mechanisms introduced in earlier sections, smart contracts can further ensure reliable policy enactment while preserving autonomy and self-organization.

(A) Decentralized supervisors

Decentralized supervisory mechanisms can enhance collective performance and improve large-scale decision-making by leveraging global information to supervise robot actions. A canonical application is *task allocation* (Figure 3.9), where achieving efficient allocations through purely decentralized, bottom-up mechanisms remains challenging.

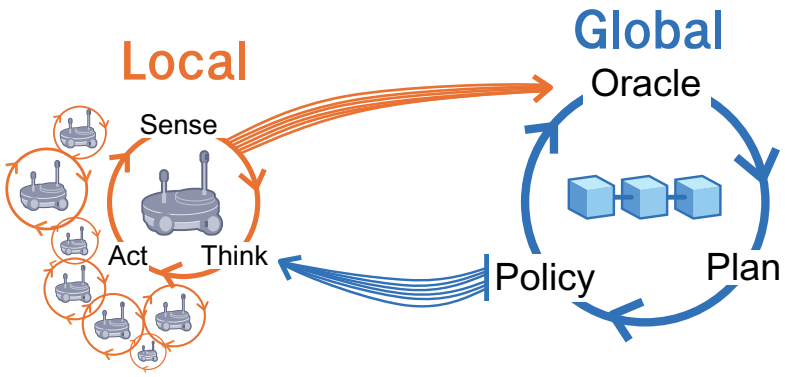


Figure 3.8: Hierarchical control of a robot swarm using smart contracts. The logic encoded in the smart contract is generated from a top-down perspective. Robots provide information they individually sense from the environment to the Swarm Oracle, enabling secure agreements upon which the smart contract can decide on a global action plan. The resulting individual policies or behavioral incentives are then fed back into the robots' local decision processes.

Realizing the potential of decentralized supervisors raises several open questions, including how to integrate supervisor-issued commands with each robot's local control, how to mitigate delays introduced by blockchain consensus, and how to maintain scalability as the number of robots and tasks grows. These issues become even more pronounced in heterogeneous or dynamic environments, where frequent online re-evaluations are necessary. In these scenarios, classical task-allocation algorithms often become combinatorial and intractable for real-time execution (Gerkey and Matarić 2004), making their implementation in smart contracts impractical as it would limit scalability.

In Chapter 7 – Section 7.2, we instantiate a *decentralized supervisor* for real-time foraging. Instead of assigning robots to specific foraging sites, the contract determines a maximum quota of foragers based on each site's quality, while robots individually signal their interest in a site by submitting transactions. This design minimizes the on-chain computational footprint by avoiding centralized planning and the need to maintain global knowledge of robot preferences. At the same time, it strikes a balance between individual autonomy and the performance benefits of top-down guidance, demonstrating good *scalability*. However, the static nature of smart-contract logic leads to performance that depends on environmental factors (Pacheco et al. 2022†). Allowing supervisory contracts to adapt their policies by learning from the environment or the robots could potentially improve system *flexibility*.

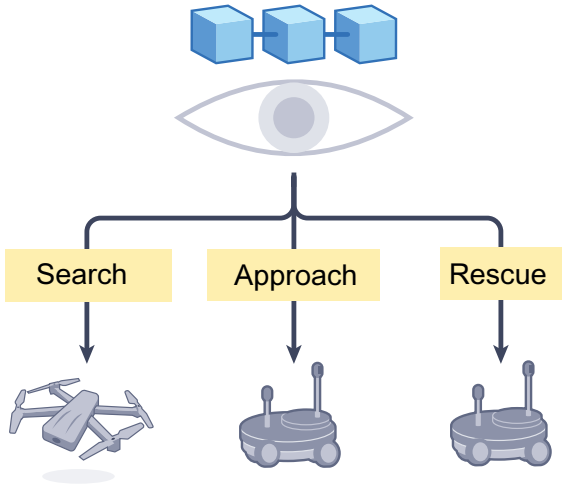


Figure 3.9: Decentralized supervisors implemented as smart contracts. Image adapted from Dorigo et al. (2024†).

(B) Group-level behavioral switches

Classical decision-making algorithms in swarm robotics (Montes de Oca et al. 2011, Reina et al. 2014b, Valentini et al. 2015b) lack mechanisms for robots to track the swarm's global state after convergence, which is essential for enacting coordinated behavioral changes. Peer-to-peer consensus processes can be slow and may fail to produce a unified decision when individual robots reject or override the inferred global state (Zakir et al. 2022), leading to delayed or fragmented system-level responses.

The deterministic logic encoded in smart contracts provides a powerful alternative for situations where ambiguity or slow convergence is unacceptable, such as energy or safety-critical situations. Smart contracts can enforce global behavioral switches that every robot must follow. Gupta et al. (2024†) demonstrated this capability in an emergency-response simulation, where robots were required to collectively commit to a fire-mitigation task as a fragmented response would result in mission failure. In this scenario, the blockchain triggered a swift, coordinated response across all robots while avoiding the limitations of a centralized controller.

Group-level switches can also potentially protect swarms, and their surroundings, from excessive sensitivity to parameter changes or environmental fluctuations that trigger dangerous or unstable collective behaviors. A smart contract with a global view can detect such conditions and issue global overrides—such as disabling all robots or forcing a safe fallback mode—to prevent the swarm from entering hazardous states.

3.4 Chapter summary

The perspective developed in this chapter synthesizes the collective insights of IRIDIA researchers and collaborators accumulated over nine years of work on blockchain-based swarm robotics. It highlights both the challenges and the opportunities that arise from integrating blockchain technologies with swarm robotic systems, reflecting on the potential of this interdisciplinary exchange. Because this perspective grew alongside the research efforts carried out during the period of this thesis, it tends to align itself with its contributions. As such, it naturally provides a *structural basis for the chapters that follow.*

- Chapter 5 focuses on the challenges of hardware feasibility and scalability by demonstrating that real robots can execute blockchain operations, and analyzing system properties under different swarm sizes.
- Chapter 6 addresses the oracle problem, bridging on-chain decision logic with the physical nature of robots, their environment, and interactions.
- Chapter 7 explores the opportunities identified in this chapter through a series of practical case studies, including foraging supervision, achieving data consistency in SLAM and federated learning in swarm robotics, and market-based mechanisms to incentivize honest information exchange among foraging robots.

The ideas presented in this perspective resonate with, and are further contextualized by, contemporary perspectives from other scholars exploring the interface between blockchain and robotics (Aditya et al. 2021, Castelló Ferrer 2023, Peña Queralta et al. 2023, Thakur et al. 2023). Together, they demonstrate the growing interest in leveraging distributed ledgers, smart contracts, and cryptoeconomic mechanisms to enhance coordination, robustness, and autonomy in robotic collectives.

Chapter 4

General Methods

This chapter presents the general methods and technical foundations that support the contributions presented in the remainder of the thesis. It focuses the shared experimental infrastructure that is used across chapters, while contribution-specific methods or parameter choices are described in their respective chapters.

In Section 4.1 we introduce the Pi-puck robotic platform used in all physical experiments, summarizing its sensing, communication, and computation capabilities.

In Section 4.2 we provide a technical overview on Ethereum, the blockchain protocol used throughout the thesis, presenting its basic functioning and structural elements—accounts, global state, transactions, blocks and the blockchain itself. In Section 4.3 we describe the permissioned consensus mechanism used to generate and agree on Ethereum blocks—Proof-of-Authority—and specify the corresponding block-production, validation, and conflict resolution rules. In Section 4.4 we detail the robots' communication, in particular how the blockchain is synchronized under mobile and intermittent connectivity. We explain our dual-layer setup: short-range infrared peer-discovery, and direct peer-to-peer Wi-Fi tunnels to synchronize blocks and transactions. In Section 4.5 we introduce the Solidity smart contracts used to coordinate swarm tasks. In Section 4.5.1 we summarize recurring contract design patterns used across the thesis, and in Section 4.5.2 we provide an overview of their task-specific instantiations. In Section 4.6 we present the Ethereum–ARGoS interface (`geth-argos`), which couples ARGoS robot entities with containerized Ethereum nodes to enable scalable blockchain-based swarm robotics studies, including the contributions developed in this thesis.

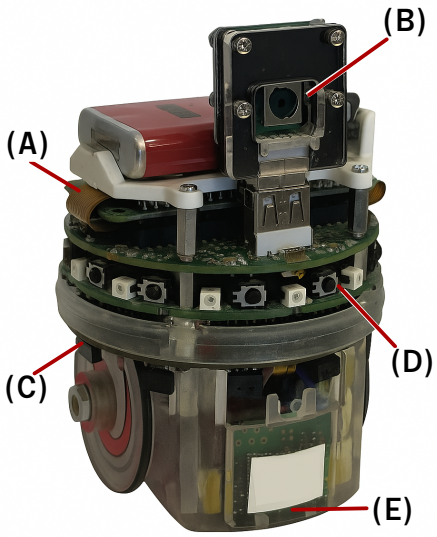


Figure 4.1: The Pi-puck robot. A differential drive robot equipped with (A) a Raspberry Pi Zero W with Wi-Fi capabilities, (B) a front-facing camera, (C) a ring of eight infrared sensors used for obstacle avoidance, (D) a range-and-bearing board that allows robots to communicate with local neighbors and (E) ground sensors which measure the reflectivity of the floor.

4.1 Pi-puck robots

All experiments in this thesis were conducted using swarms of up to $N = 24$ *Pi-puck robots*, which are e-puck robots (Mondada et al. 2009) augmented with an extension board (Millard et al. 2017). This extension allows the robots to be controlled by a Raspberry Pi Zero W, featuring a 1 GHz processor, 512 MB of RAM, and a 16 GB SD card for data storage. The board additionally equips the robots with Wi-Fi connectivity and sufficient computational power to run a Linux-based system. The robots are also equipped with a range-and-bearing module (Gutiérrez et al. 2009a), a ground sensor that allows them to detect the color of the floor (used in Chapter 5) and a front-facing camera (used in Chapter 6). The robot is depicted in Figure 4.1, and Table 4.1 specifies its hardware components and their utility for our setup.

4.2 Ethereum blockchain

Ethereum is the blockchain platform that was used throughout this thesis. Proposed by Buterin (2014), it builds upon and generalizes Bitcoin's core ideas to generate a blockchain envisioned as a "world computer" by enabling smart contracts—programs

Table 4.1: Hardware components of the Pi-puck robot.

Component	Description
(A) Raspberry Pi Zero W	Linux-capable processor that runs the blockchain and controls the remaining components via I2C
(B) Front Camera	Used to measure the RGB values of environment features in Chapter 6
(C) Infrared sensor ring	Detects obstacles; used for obstacle avoidance
(D) Range-and-bearing board	Enables local infrared-based communication; used to exchange robot identities and cryptographic hashes
(E) Ground sensor	Detects the reflectivity of the color; used to distinguish between black and white tiles in Chapter 5

written in a Turing-complete language and executed collectively by the network.

Several characteristics make Ethereum particularly strong candidate for deployment within robot swarms

- Support for *smart contracts*, which provide a generic interface to deploy programs to coordinate swarm tasks;
- A *consensus protocol* that avoids the high energy consumption of proof-of-work and can operate with limited connectivity among robots;
- Compatibility with *resource-limited hardware*, enabling deployment on the Pi-pucks or other swarm robotics platforms.

As of October 2025, the public Ethereum network uses energy-efficient proof-of-stake consensus, following its migration from proof-of-work in September 2022. For permissioned deployments, Ethereum clients also support proof-of-authority consensus, which provides predictable block times, low computational overhead, and straightforward consensus rules. These properties make it robust to network partitioning and well aligned with the communication and mobility constraints inherent to swarm robotics. All of the works presented in this thesis used proof-of-authority.

Although alternative blockchains and consensus protocols could have been selected, Ethereum—being the first blockchain to introduce smart contracts—provides a mature and well-documented implementation. In Chapter 2, we presented alternative protocols used by other researchers, and in Chapter 3, we discussed several advantages and trade-offs presented by different protocol design choices in terms of hardware requirements and scalability.

The remainder of this section explains the basic functioning of the Ethereum blockchain and presents its building blocks: **Accounts**, **global state**, **transactions**, **blocks** and the **blockchain**.

(A) Accounts

An Ethereum account (Figure 4.2) is identified by a public address and is associated with a balance of ether (Ethereum's native cryptocurrency). There are two types of accounts: **Externally Owned Accounts (EOAs)**, which are controlled by private keys and represent *users* (or the robots in our case), and **Smart Contract Accounts (SCAs)**, whose public address is mapped to *executable code and a persistent state*. Both types can send and receive ether, but only contract accounts store and execute code.

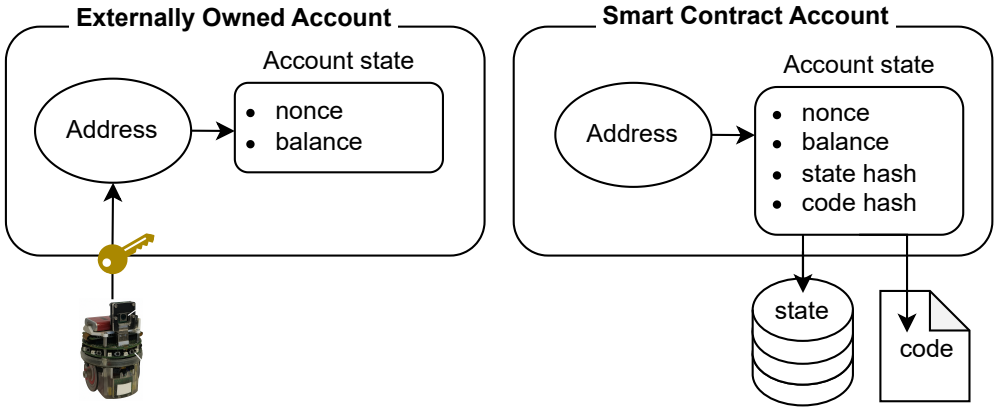


Figure 4.2: Ethereum account types. Ethereum accounts are of two types: EOAs, controlled by users or robots, and SCAs, which are smart contract accounts associated with a code-state pair.

(B) Global state

The global state (Figure 4.3) is a snapshot that maps every public address to its current account data, including user balances and the code and storage of smart contracts. Simply put, the goal of blockchain technology is to ensure that every node agrees on the same exact global state.

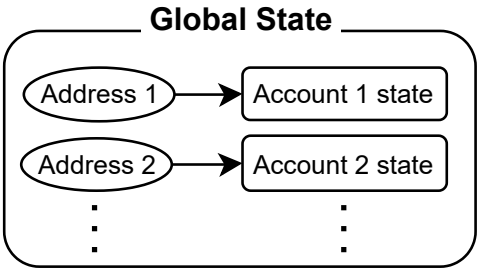


Figure 4.3: Ethereum global state. A map between public addresses and the state of SCAs and EOAs.

(C) Transactions

Transactions (Figure 4.4) are cryptographically signed instructions that modify the global state by:

(i) transferring ether between accounts;

(ii) creating new contracts;

(iii) executing functions in existing contracts to update their state.

Each valid transaction transforms the state into a new one, so Ethereum can be understood as a sequence of state transitions from an initial (genesis) global state to the current one.

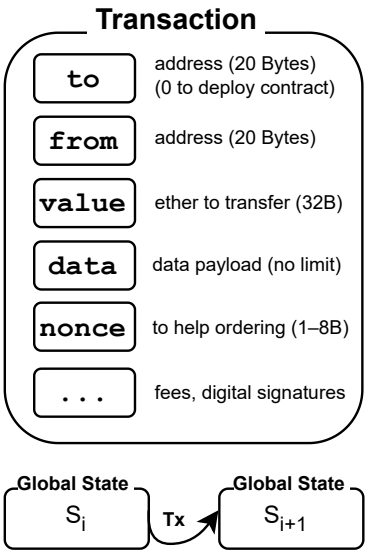


Figure 4.4: Ethereum transaction. State-changing operation that modifies user balances or contract states.

(D) Blocks

Blocks (Figure 4.5) define a combined state transition that results from the ordered execution of multiple transactions. Structurally, a block consists of the *header* and the *body*. The *block body* stores the list of transactions, sequenced by the creator of the block (the block producer). The *block header* contains metadata, such as block number, timestamp, the cryptographic hash of the parent block, the root hash of the global state after processing its transactions, and the block producer address. This metadata (508 bytes) suffices to guarantee the block is valid and connected to the blockchain history (i.e., nodes don't need the body to check whether the block follows consensus rules).

(E) Blockchain

A blockchain (Figure 4.6) is a cryptographically linked sequence of blocks. Each block references its predecessor through the parent hash stored in its header, creating an easy-to-validate chain. This design guarantees immutability: altering any block would require recalculating every subsequent hash, thereby violating consensus. The blockchain maintains a consistent and transparent record of all state transitions. The network's consensus protocol ensures that all participants agree on a single version of the blockchain.

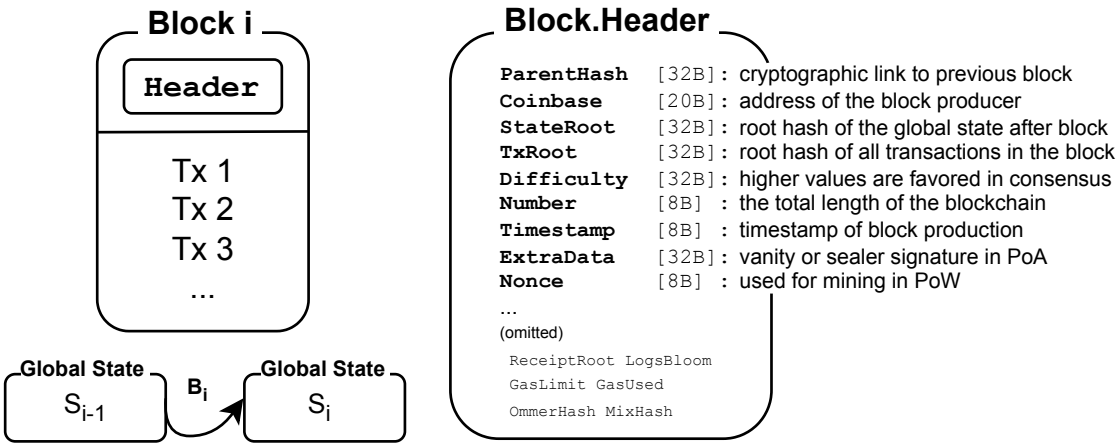


Figure 4.5: Ethereum block. Block B_i represents a transition from global state S_{i-1} to S_i. This transition results from the ordered execution of the transactions in the *block body*. The *block header* contains cryptographic summaries of this transition, enabling nodes to verify consensus properties (e.g., timestamp, proposer, size limits) and to ensure data integrity without downloading the full body. In particular, the `StateRoot` commits to the post-execution global state, while the `TxRoot` commits to the ordered list of transactions.

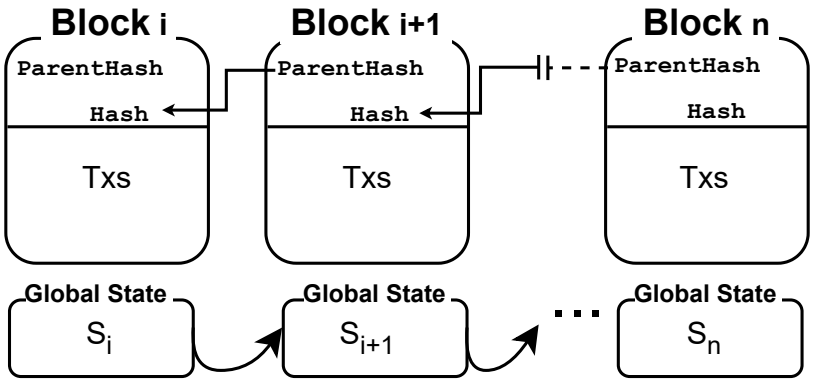


Figure 4.6: Ethereum blockchain. The blockchain is a linked sequence of blocks, where each block stores the `ParentHash` of its predecessor. As new blocks are appended, this structure forms a tamper-evident and immutable history of global state transitions: altering any past block would change its hash, therefore breaking the chain.

4.3 Consensus protocol

In a blockchain system, the consensus protocol determines which nodes may produce blocks and ensures that, in the presence of conflicting blocks, the network converges on a single canonical blockchain. In this thesis, we use proof-of-authority because it is: (i) *Byzantine fault-tolerant* up to 50% malicious nodes; (ii) *permissioned*, avoiding the energetic costs of proof-of-work; (iii) *probabilistic and gossip-based*; and (iv) *partition-tolerant*, halting only if more than half of the nodes are unavailable.

In proof-of-authority, a list of nodes authorized to produce blocks (called "sealers") is recorded in the genesis block—the first block of the blockchain. The sealers take turns proposing blocks, which are then broadcast to the network and validated by every other node. If the sealer whose turn it is is not available, another takes its' place

We deployed the Clique implementation of proof-of-authority (Szilágyi 2017), which operates as follows. For each block, a preferred sealer is designated in a round-robin schedule. If the preferred sealer produces and signs the block, the signature is considered *in-turn*; otherwise, if the preferred sealer is unavailable or fails to produce a block within the allotted time, another authorized sealer may produce the block *out-of-turn*.

Consensus rules The header of a valid block must follow these rules:

- it must be linked to a valid parent block and have the right block number;
- its timestamp is at least B_p (block period) seconds after the parent block;
- its sealer must be authorized and provide a correct signature;
- its sealer must not have signed another block in the last $\lfloor \frac{N}{2} \rfloor + 1$ blocks;
- its difficulty is 2 when signed in-turn, and 1 otherwise.

Block validation and propagation The sealer broadcasts a signed block header to its peers in the network (approximately 508 bytes). Each peer checks that the block header follows the rules above, and if not, the block is discarded. Then, if the block does not conflict with another (i.e., does not have the same parent), the node adds it to the blockchain, executes its transactions, and broadcasts it to peers. If conflicting blocks occur, the node chooses according to the *strongest chain* rule, that is, the chain with the highest cumulative block difficulty.

Dynamic sealer set In this thesis, swarms were composed of fixed numbers of robots during experimental trials (between 5 and 24 physical robots, and up to 120 in simulation). Clique, however, supports adding and removing sealers through a built-in voting mechanism.

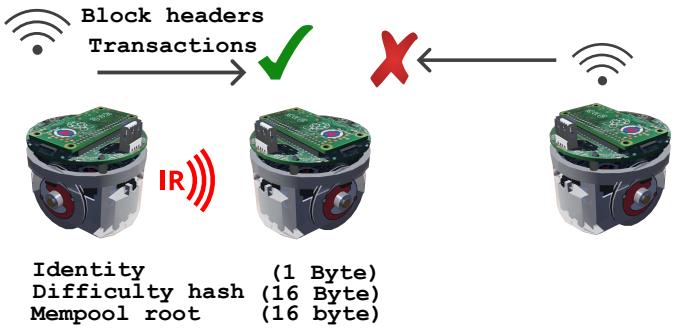


Figure 4.7: Two-layer communication protocol. In the figure, the robot in the center received an infrared signal from the robot on the left; therefore, it accepts incoming block headers and transactions via Wi-Fi. The connection attempt from the robot on the right is denied, since it is out of infrared range.

4.4 Communication protocol

Communication among robots occurs in two layers: short-range infrared broadcasts, used to handle *peer discovery*; and direct Wi-Fi tunnels, used to *synchronize* the peers' mempools and blockchain. In this way, scalable infrared broadcasts are used to signal when to synchronize, while interference-prone Wi-Fi is used only if new information is available. This protocol is represented in Figure 4.7.

Peer discovery Robots listen for infrared signals through their range-and-bearing board at ∼30 B/s (Gutiérrez et al. 2009a). This can be used to broadcast payloads such as the robots' public identities, data hashes, and digital signatures (similarly to modern smartphones' NFC payments). Using short-range signals to discover peers adds a security layer as it protects the swarm from remote attacks since robots do not accept incoming information until the infrared greeting is acknowledged. Throughout this thesis, different communication ranges—controlled by tuning the power of infrared signals—were set according to the application scenario and research objectives.

Blockchain synchronization Robots block all incoming Wi-Fi connections until they receive infrared peering requests. Once a suitable peer is discovered via infrared, in particular, if that peer is advertising new blocks or transactions, the robot opens a direct Wi-Fi tunnel to synchronize the blockchain following the consensus rules and Ethereum client software implementation.

Improvements Early versions of the system followed the mechanism described in Pacheco et al. (2020†) and Strobel et al. (2023†): robots would add neighboring peers after receiving their ID via infrared, and would remove them after a 2 s grace period. While functional, this strategy often resulted in robots maintaining several

Table 4.2: Typical payload sizes and communication rates.

Item	Size / Rate	Notes
Robot identity	1 B	Integer used to identify the robots
MD5 hash	16 B	To detect if peers have new blocks or transactions
Block header	508 B	Default Ethereum header, including unused fields
Basic transaction	100–300 B	Transfers or contract calls without data payload
Transaction data	32 B	Chapter 5
	224 B	Chapter 6
	16.8 KB	Chapter 7 – Section 7.3
Range-and-bearing	∼30 B/s	Exchange of IDs and hashes
Wi-Fi (802.11 b/g/n)	∼1–10 MB/s	Transfer of block headers and transactions

simultaneous Wi-Fi tunnels, thereby leading to congestion.

To improve scalability, we extended the infrared exchange to include not only the robot ID (1 Byte), but also two compact summaries of each robot's blockchain state: the MD5 hashes of the chain difficulty and of the mempool root (16 Bytes each). With this additional information, a robot only initiates a Wi-Fi tunnel when a neighbor has a stronger chain or possesses novel transactions. This illustrates how blockchain data structures—specifically, cryptographic summaries of local states—can be exploited to improve communication efficiency in robot swarms. Table 4.2 shows the data sizes of the structures exchanged among the robots, and the communication rates of the available mediums.

Considerations In a proof-of-authority blockchain, the block period B_p determines how frequently new blocks are created and, consequently, directly influences how frequently robots must synchronize with others. In order to achieve efficient operations, a block should be disseminated to most robots before B_p has elapsed; otherwise, sealers may produce conflicting blocks that generate blockchain forks. These forks lead to wasted communication and processing power, as the robots reset their state to the canonical chain and discard the blocks in the forks. Although proof-of-authority consensus is robust to temporary network partitions, maintaining good connectivity within the swarm reduces the likelihood of forks. A detailed analysis of performance when the swarm's connectivity changes is presented in Chapter 5.

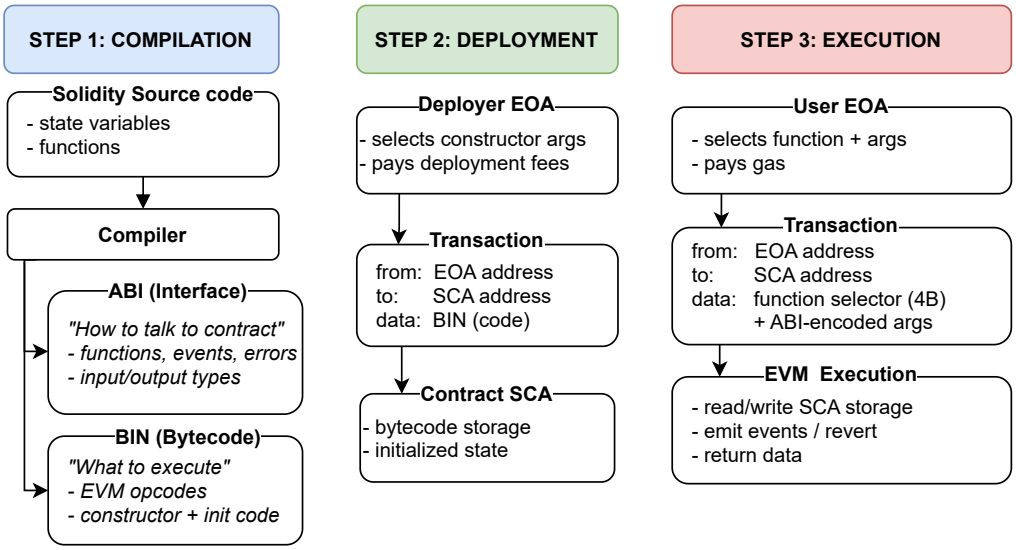


Figure 4.8: Ethereum smart contracts. Step 1) Solidity smart contracts compile into two artifacts: the ABI, which tells users how they can interact with the contract (e.g., execute functions or subscribe to events), and the BIN bytecode. Step 2) Deployment of the contract occurs via an on-chain transaction that contains the BIN as the data payload, generating a new contract address and initializing its state. Step 3) Users use the ABI (hosted and exchanged outside the chain) to form structurally valid transactions that interact with the smart contract.

4.5 Solidity smart contracts

Solidity is Ethereum's primary high-level programming language for developing smart contracts. In contrast to Bitcoin's Script, where operations are deliberately restricted, Solidity is Turing-complete and supports general-purpose contract logic. Contract execution must nonetheless be deterministic so that all nodes compute the same states and consensus remains possible. Solidity provides user-defined types, modular libraries, inheritance, and event emission. Solidity source code compiles to Ethereum Virtual Machine (EVM) bytecode and exposes an Application Binary Interface (ABI), which specifies how to interact with the contract via transactions, as illustrated in Figure 4.8.

Each contribution in this thesis is associated with a contract programmed in advance and deployed prior to the experiments to coordinate the swarm during the corresponding task. Nevertheless, one could also envision scenarios in which robots autonomously deploy contracts—e.g., to instantiate temporary sub-swarm hierarchies, or to self-organize around emergent tasks discovered in the environment.

The remainder of this section introduces the design patterns used by contracts in this thesis and provides an overview of their task-specific instantiations.

4.5.1 Design patterns

Across the thesis, smart contracts implement a set of patterns that are common in blockchain systems.

Input aggregation Robots submit local values through transactions; the contract filters and aggregates them, emitting an updated shared state (e.g., a collective estimate, database or model). This pattern underpins consensus achievement in Chapters 5 and 6 as well as collaborative mapping and federated learning in Chapter 7.

Token-based participation Token-based mechanisms are used to mitigate Sybil attacks and transaction spam by making participation and data contributions towards collective tasks costly or rate-limited. In Chapter 5, contributing data towards consensus costs 40 tokens, which are refunded when robots' submissions are not flagged as outliers. In Chapter 6, submitting data costs $K\,t_i$ tokens, where t_i is robot i's current token balance and K controls the maximum number of parallel consensus tasks (i.e., it constrains how much of its balance a robot can commit across concurrent tasks). In Chapter 7 – Section 7.4, the mapping smart contract uses two token types: (i) authorization tokens, which grant participation rights and can be revoked upon Byzantine behavior, and (ii) reward tokens, which serve as metric of robots' performance. New tokens are generated and distributed to robots either on a fixed schedule (Chapter 5), or as rewards for contributing towards collective tasks Chapters 6 and 7.

Round-based updates Definitive updates to the shared state occur when a quorum of submissions is reached, reducing the influence of temporary partitions or Byzantine robots. In Chapter 5, a round finishes when N (swarm size) transactions are submitted, regardless of robot identity; in Chapter 6, each round requires that robot submits a total of $\frac{2}{3}K$ of the circulating reputation tokens; in Chapter 7, the federated learning smart contract requires a fixed quorum of seven participants (to replicate the baseline study); and the mapping one requires three distinct robots to validate each others' loop closures (coordinate transformations linking poses along robots' trajectories) by forming geometrically valid triangles.

Reputations and weighted participation Depending on the outcomes of the round-based updates, robots can either gain or lose reputation tokens. In Chapter 5, robots with more tokens can afford to submit their values more frequently; while in Chapter 6, tokens deposited alongside the values are used as weights during aggregation of the values.

Crypto token incentives By aligning token incentives with truthful information and active participation, we generate a base mechanism for token economies that neutralizes the impact of Byzantine robots and promotes trust. In the information market (Chapter 7 – Section 7.5), robots are rewarded by providing useful and correct information to peers, influencing the level of trust these peers place in future exchanges.

4.5.2 Task-specific instantiations

This thesis implemented a total of 5 smart contracts for different applications. These smart contracts are described in more detail in their respective chapters, and are available as open-source code (Pacheco and Strobel 2020†, 2022†). Each one of these instantiations can be seen as a oracle problem and corresponding solution: the robots gather and process data, which is then aggregated into a shared database, using a smart contract. In Chapter 7, this shared data is then used to coordinate the robots during foraging, learning and mapping tasks.

Collective sensing (Chapter 5) This smart contract secures consensus in a robot swarm by combining outlier detection with a crypto-token economy. To protect against Sybil attacks and double spending, robots have to pay a token deposit to submit their sensed values. Robots whose are flagged as outliers lose their deposits, while others are rewarded. Initial funds are granted through a universal basic income (UBI) mechanism, given to all robots.

Swarm Oracle (Chapter 6) This smart contract generalizes the previous mechanism to arbitrary consensus spaces (e.g., multi-dimensional or discrete) by aggregating the sensor values submitted by robots into *proposals* via a clustering algorithm. Robots independently verify each proposal and issue confirmation or rejection votes, implementing a Byzantine fault-tolerant algorithm that tolerates adversarial behavior by up to one third of the robots. A token-based reward mechanism gradually distinguishes faulty robots from malicious ones and weights their influence in consensus achievement, enabling automatic recovery from both faults and coordinated attacks.

Real-time foraging supervisor(Chapter 7 – Section 7.2) This smart contract acts as a *decentralized supervisor* that aggregates newly discovered resource patches into a shared database and computes assignment policies (e.g., limiting the number of foragers per patch, or prioritizing high-quality patches). Robots send transactions to signal their intent to forage at specific sites, enabling the contract to coordinate swarm foraging in real time.

Federated learning without server (Chapter 7 – Section 7.3) Robots submit locally trained neural network models via transactions to the smart contract. Once

a quorum of models is reached, the contract aggregates them—weighted by the amount of local data used for training—to produce a shared global model, which robots then retrieve to initialize their next local training round. Tokens limit robots' participation rates, mitigating Sybil attacks and providing a reputation-like measure of robot's model quality.

Loop closure validation in Swarm-SLAM (Chapter 7 – Section 7.4) Robots submit loop closures (geometrical transformations between points along their trajectories) to the smart contract. The contract accepts loop closures that form geometrically valid triangles between at least three robots, preventing a malicious robot from injecting incorrect loop closures towards another robot's trajectory, which would introduce false constraints in the optimization of the collective map. *Note:* This contract was implemented in Python, not Solidity.

4.6 ARGoS simulation interface

Our simulation interface is publicly available and documented in an online repository (Pacheco and Strobel 2022†).[1] Over the years spanning this thesis, it has undergone several improvements to ultimately support simulations with up to 120 robots (Strobel et al. 2023†). The interface integrates multiple software components: the ARGoS simulator (Pinciroli et al. 2012), the e-puck plugin (Garattoni et al. 2021), the ARGoS Python wrapper (Hasselmann et al. 2021†), the Ethereum Go-client,[2] and Docker (Merkel 2014). Together, these modules form the `geth-argos` framework illustrated in Figure 4.9. This framework has supported the research of four PhD students and several MSc students, and has become a central tool for developing, testing, and benchmarking blockchain-enabled robot swarms.

The interface operates as follows:

- When initiating an experiment with N robots, a cluster of N Docker containers running Ethereum's Go client (`geth`) is launched. Subsequently, ARGoS launches the experimental environment along with N robot entities.
- After building the ARGoS environment, most user-facing scripts are in Python: the environment loop functions and the robot controllers with Ethereum interactions. The smart contracts are written in Solidity.
- The ARGoS C++ user-facing components (robot controllers and loop functions) are exposed to Python scripts through `Boost.Python`.[3] The implemen-

[1] `https://github.com/teksander/geth-argos`
[2] `https://geth.ethereum.org/`, accessed November 2025
[3] `https://www.boost.org/`, accessed November 2025.

tation of this C++/Python wrapper is publicly available online (Hasselmann et al. 2021†).

- Communication with the `geth` API is facilitated by the Python library `web3.py`.[4] This library provides essential blockchain-related functions for the robots, including `add_peer`, `send_transaction`, and `get_block`.
- Directly exposing `web3.py` to the robots introduced performance issues, such as simulation throttling. To mitigate this problem, each Docker container hosts an `RPyC`[5] server, which supports asynchronous queries and improves communication efficiency.

4.7 Chapter summary

This chapter presented the general tools underpinning the work in this thesis.

We first introduced the Pi-puck robot platform, detailing its sensing, communication, and computation capabilities, and explained how it supports blockchain execution in swarms of up to $N = 24$ physical robots. We then described the structure and functioning of Ethereum, including the notions of accounts, global state, transactions, blocks, and the blockchain, and motivated our choice of a permissioned proof-of-authority consensus protocol.

We outlined the two-layer communication architecture in which short-range infrared broadcasts are used for peer discovery and exchange of compact blockchain summaries, while Wi-Fi tunnels are opened selectively to synchronize blocks and transactions. We discussed how this design reduces congestion and exploits blockchain data structures to improve communication efficiency.

We introduced the basic functioning of Solidity smart contracts and the set of recurring design patterns used throughout the thesis. We then presented five task-specific contract instantiations, which will be detailed and discussed in the coming chapters.

Finally, we described the `geth-argos` simulation interface, which couples the ARGoS simulator with Dockerized Ethereum clients to enable scalable experiments, showcased with up to 120 robots in Chapter 5. Together, these hardware and software frameworks provide a unified experimental pipeline for developing, testing, and evaluating blockchain-enabled swarm robotics systems in the subsequent chapters.

[4] `https://web3py.readthedocs.io/`, accessed November 2025.
[5] `https://rpyc.readthedocs.io/`, accessed November 2025.

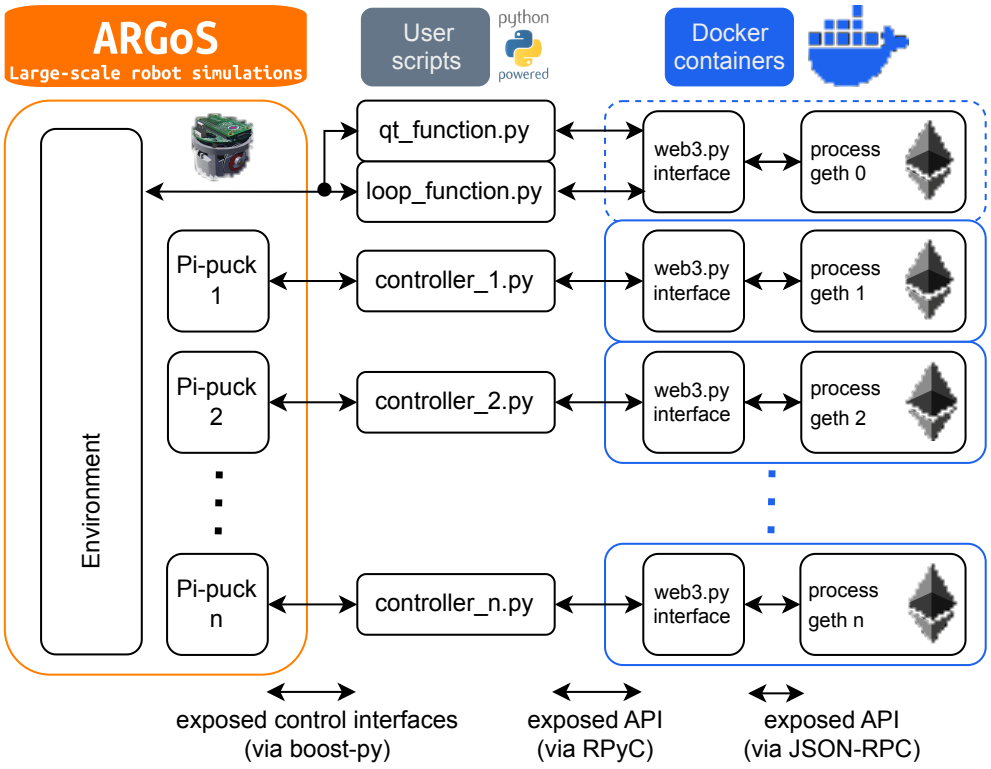


Figure 4.9: The interface between ARGoS and Ethereum. In the `geth-argos` interface, each ARGoS robot instance is linked to an Ethereum node running in a Docker container. Each container is allocated limited CPU and RAM to match the constraints of the physical robots. The robot control interface and the Ethereum client API are exposed as Python modules, allowing users to control the robots using only Python programs.

Chapter 5

Proof of Concept: Scalability and Hardware Requirements

This chapter integrates contributions from two works: Pacheco et al. (2020†), which presented the first blockchain deployment in a swarm of 10 Pi-pucks; and Strobel et al. (2023†), which extended the physical experiments to swarms of up to 24 robots and to large-scale simulations with up to 120 robots.

In the experiments, the robots were tasked with estimating the proportion of white tiles in checkerboard environments, a classical consensus-achievement scenario in swarm robotics (Valentini et al. 2017). At the time of publication, our work was the first deployment of a blockchain in a robot swarm that relied solely on local communication. As such, our work was a key proof-of-concept, demonstrating to the swarm robotics community the feasibility of using a blockchain to secure consensus achievement.

This chapter focuses on the fundamental and technical aspects of deploying blockchain-based robot swarms. Concretely, there are six main objectives:

1. Demonstrate that resource-constrained robots can *meet the hardware requirements* to maintain a permissioned blockchain (feasibility).

2. Characterize the *hardware scalability* by quantifying the increases in CPU, storage, and network usage by the robots over time and across swarm sizes.

3. Study how the blockchain's permissioned consensus *scales and performs* in the highly dynamic and partitioned swarm robotics networks under different swarm sizes and arena sizes.

4. Demonstrate that smart contracts can be used to *protect consensus achievement* from the presence of Byzantine robots.

5. Characterize the *robustness, scalability, and generality* of the Byzantine protection under varying swarm sizes, arena sizes and configurations, and the quantity and type of Byzantine robots.

6. Critically discuss the advantages, vulnerabilities, and challenges of the approach, providing guidelines for future work.

The remainder of this chapter is structured as follows. Section 5.1 introduces and motivates the research. Section 5.2 describes the environment and task, the smart contract, the crypto token economy, the robot controllers, the Byzantine robot behaviors, and the experimental setup. Section 5.3 presents the experimental results focusing the systems' robustness, scalability, generality and resilience. Finally, Section 5.4 concludes the chapter, discussing the approach and its vulnerabilities, and indicating directions for future research.

5.1 Introduction

In real-world deployments, robot swarms will face several security challenges that are rarely taken into consideration in swarm robotics research (Higgins et al. 2009). In particular, the presence of *Byzantine* robots, that is, malfunctioning, faulty, or malicious robots, can potentially lead to a discrepancy between the *designed* and the *actual* behavior of a swarm. Strobel et al. (2020) initially proposed the use of blockchain technology to greatly limit, in a fully decentralized manner, the impact of Byzantine robots on the overall swarm behavior.

Beyond the Bitcoin's blockchain use case as a decentralized *ledger* for financial transactions, blockchains were also introduced as decentralized *computing platforms*. In the Ethereum blockchain, network participants can run programming code on the blockchain and agree on the outcome of the programs (Buterin 2014). These decentralized programs have been shown to be useful in robot swarms, particularly for addressing security issues (Strobel et al. 2020).

However, multi-robot systems interfacing with blockchain and smart contracts have only been studied either on simulated robots (Strobel et al. 2020, Strobel and Dorigo 2018), or on robots interfacing with externally hosted blockchains (see Chapter 2 for a comprehensive review). The work here presented was therefore the first implementation of this technology in a physical robot swarm where each robot hosts the blockchain independently.

Achieving this is a crucial step towards convincing the robotics community of the feasibility of blockchain and smart contract technology in robot swarms. However, moving from simulated to real robots is not straightforward and involves, among other things, the choice of adequate robot platforms, communication protocols, and blockchain frameworks. In previous works addressing this topic, many questions

whose answers would be of paramount importance for a successful implementation on real robots were not addressed. Examples are:

- Which lightweight robot platform can provide the requirements for running a blockchain framework?
- Which blockchain consensus protocols are appropriate for robot swarms?
- Which communication methods should be used?
- How does blockchain synchronization perform as the swarm size increases?

In this chapter, we provide a first answer to the above questions by benchmarking the experimental setup described in Chapter 4. This setup utilizes Pi-puck robots (Millard et al. 2017), which maintain a permissioned proof-of-authority blockchain and communicate through a mobile ad hoc network. The Pi-pucks are a reasonable choice due to their relatively low cost and Raspberry Pi onboard computer. The proof-of-authority protocol is adequate to the Pi-pucks capabilities, providing the desired security and consensus guarantees on the state of smart contracts. Finally, the dual-layer communication scheme—where Wi-Fi communication is only enabled after a local infrared exchange of IDs—reduces interference and enhances security.

These choices are overall consistent with the standard swarm robotics requirements of peer-to-peer communication, simple and inexpensive robots with on-board computations, and decentralized consensus (Brambilla et al. 2013, Dorigo et al. 2014, Garattoni and Birattari 2016).

5.2 Methods

The tools employed in this chapter and throughout the thesis are described in Chapter 4, including the robots, the simulation interface, the Ethereum-based blockchain, and the communication and consensus protocols. The following chapter-specific parameter values were used:

- The *block period* B_p is 15 seconds (the minimum time difference between two consecutive blockchain blocks in proof-of-authority consensus).
- The *maximum communication distance* is 10 cm (in real robots this is regulated by tuning the infrared signal power).

The remainder of this section presents the methods specific to this chapter.

5.2.1 Environment and task

The robots' task is to estimate and agree on the fraction of white tiles in an environment containing black and white floor tiles. The robots perform random walks in the environment while measuring tile colors using ground sensors. In addition to noise-induced errors, some robots act as Byzantine agents that disseminate incorrect estimates to their peers. This chapter combines the results of two works, organized as two datasets: `ants2020` (Pacheco et al. 2020†) and `scirob2023` (Strobel et al. 2023†). The experimental environments used across the studies are shown in Figure 5.1, and the dimensions and layout of the experimental arenas are shown in Table 5.1.

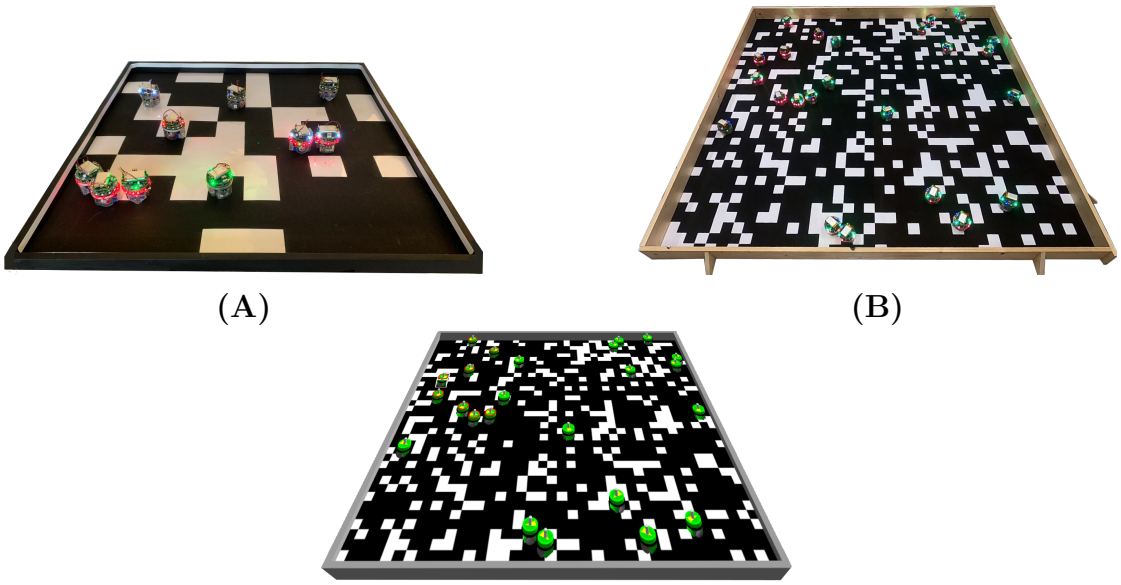


Figure 5.1: Experimental environments. (A) In Pacheco et al. (2020†) we presented the first blockchain deployment in a swarm composed of 10 Pi-pucks in a $1\,m^2$ arena, demonstrating the feasibility of the approach. **(B)** In Strobel et al. (2023†), we scaled the experiments up to 24 robots in a $3.61\,m^2$ arena, the largest of the three physical arenas in the study. **(C)** Large-scale simulations in ARGoS replicate the physical parameters in software, enabling the study of scalability with up to 120 simulated robots.

Table 5.1: The sizes of the different swarms and arenas.

Dataset	Side length (m)	Side length (tiles)	Area (m^2)	Area (tiles)	Swarm size (robots)
`ants2020` (robots)	1.00	22	1.00	484	5, 6, 7, 8, 9, 10
`scirob2023` (robots and simulation)	1.10	22	1.21	484	8
	1.55	31	2.40	961	16
	1.90	38	3.61	1444	8, 16, 24
`scirob2023` (simulation)	2.70	54	7.29	2916	48
	3.30	66	10.89	4356	72
	3.80	76	14.44	5776	96
	4.25	85	18.06	7225	120

5.2.2 Smart contract

The smart contract is responsible for aggregating the robots' estimates and for determining when consensus has been achieved. It exposes five functions through which the robots can interact:

1. `registerRobot()` is called by each robot at the start of the experiment, and is required before a robot may call any of the following functions. It increments the swarm size by one, and stores the robot's public address.

2. `sendEstimate(<localEstimate>)` lets the robots *report* their local estimates by sending 40 tokens through a transaction.

3. `askForUBI()` lets robots request the universal basic income by sending a transaction (see UBI explanation below).

4. `getEstimate()` returns the shared estimate.

5. `hasConverged()` returns whether convergence on an estimate has been achieved. In `ants2020`, it returns "true" when the absolute difference between the last two shared estimates is smaller than $\tau = 1\%$. In `scirob2023`, τ is decreased to 0.2%.

The first three functions can only be executed via transactions, since they change the state of the smart contract. The last two are call-only functions, which simply return the values from the smart contract state currently stored locally by the robot.

5.2.3 Crypto token economy

In order to report their local estimate to the smart contract, robots broadcast `sendEstimate` transactions accompanied by a fixed amount of 40 tokens. Robots can obtain tokens in two ways:

- by receiving rewards for sending estimates;
- by receiving the universal basic income (UBI).

This forms the basis of a crypto-economic system, where token incentives are provided for individual contribution, while the UBI ensures tokens are initially distributed to enable new participants.

Rewards and outlier rejection The estimates sent by robots are stored in a temporary list in the smart contract. When the list accumulates N estimates, the contract filters out any estimates that deviate from the current shared estimate by more than $\delta = 0.1\%$ (except for the very first N estimates, which are all accepted). The shared estimate is then updated as a rolling arithmetic mean of the accepted values, and the temporary list is erased. All robots that submitted an accepted estimate receive back their *40 tokens plus a bonus*, consisting of a share of the non-repaid tokens from discarded estimates.

Universal basic income The UBI is an economic mechanism built into the smart contract to distribute crypto tokens among robots. It functions as follows: at block numbers that are powers of two (that is, blocks 1, 2, 4, 8, ...), the contract transfers 20 tokens to each robot in the swarm when the robots execute `askForUBI()`. This exponential scheme ensures that every robot receives enough tokens to initially be able to send its local estimate; while over time, sending accepted estimates becomes the main means to receive additional tokens and to be able to continue participating in the experiment.

5.2.4 Robot controllers

The behavior of each robot is composed of six high-level routines, which are executed concurrently. The robot controllers and interactions with the `geth` client were implemented in Python and are available online, both in simulation and reality (Pacheco and Strobel 2020†, 2022†).

Random-walk with obstacle avoidance The robot can either move straight, rotate on the spot, or avoid obstacles. If any of the robot's eight obstacle sensors are triggered (< 5 cm), the robot avoids it by turning either left or right. Otherwise, the robot alternates between phases of straight motion and rotation, with the duration of each phase drawn from an exponential distribution, as done by Valentini et al. (2016).

Peering Robots use their range-and-bearing board to exchange IDs with nearby robots (< 10 cm). After receiving a new ID, the robot establishes a Wi-Fi connection to exchange blocks and transactions. If no IR messages are received within 2 seconds, the Wi-Fi connection is closed.

Estimation Every second, each robot samples the floor using its ground sensor and classifies it as either black or white based on a fixed threshold. The robot then updates its local estimate of the proportion of white tiles in the environment by dividing the number of white classifications by the total number of readings.

Reporting Each robot sends its local estimate and corresponding token deposit to the smart contract through transactions. Reporting takes place every 45 seconds in `ants2020`. In `scirob2023`, reporting occurs as soon as enough tokens are available.

Block production Blocks are produced according to the proof-of-authority protocol, described in Chapter 4. The preferred block sealer broadcasts a new block $B_p = 15$ s after the previous one. If other robots do not receive this block within a time window, they produce competing blocks.

Blockchain synchronization Each robot runs an instance of the Ethereum client `geth`. When two robots establish a peer connection, they exchange blocks and transactions according to the Ethereum synchronization protocols.

5.2.5 Byzantine robots

In the distributed-systems literature, Byzantine faults denote arbitrary, erratic, and potentially adversarial behaviors that cannot be attributed to a specific fault source (Lamport et al. 1982). In this chapter, we analyze and mitigate the impact of Byzantine robots that submit incorrect local estimates to the smart contract. This type of Byzantine fault can arise from a variety of underlying reasons, including:

- Impaired robot mobility, leading to biased sensor estimates.
- Misalignment or displacement of the robots' ground sensor.
- Sensor drift or inconsistent calibration.
- Intermittent communication errors between sensors and processors.
- Software faults, failed updates, or malicious tampering.

In our experiments, the default Byzantine robot behavior consists of persistently submitting an estimate of 0% to the smart contract. To assess the generality of our results, we additionally consider two alternative instantiations of this fault model: (i) submitting either 0% or 100% at random, and (ii) submitting random values between 0% and 100%.

5.2.6 Experimental setup

(A) Initialization and termination

Before each experimental trial, the robots are randomly distributed in the arena (in physical experiments, this is done manually by the experimenter). The choice of which robots are Byzantine is randomly determined by a script, both in reality and in simulation. Then, a monitoring computer sends a start signal to begin the trial. The trial is stopped once all robots receive `true` when querying `hasConverged()`, at which point they turn on their green LEDs.

Genesis block All robots must initialize the blockchain using the same genesis block, which encodes the consensus rules, list of block sealers, and the initial state. When the swarm size changes, the genesis block is updated with the new list of sealers, and uploaded to each robot.

Smart contract deployment In `ants2020`, the smart contract is deployed by emitting a transaction before starting the experiment, which is then executed in block 1. In `scirob2023`, the smart contract is hardcoded in the genesis block.

Convergence criterion In `ants2020`, `hasConverged()` returns `true` when the absolute difference between the last two shared estimates is smaller than 1 %. In `scirob2023`, this threshold is tightened to 0.2 %, allowing experiments to run longer and achieve higher quality results.

(B) Independent variables

Each experimental configuration is characterized by a change in one or more of the following variables.

Number of Byzantine robots To study the *robustness* of our approach we vary the number of Byzantine robots f while keeping the overall swarm size constant. This ensures that physical parameters, such as the average connectivity, remain consistent and do not influence the results.

Number of robots Changing the swarm size N allows us to study two key properties: *scalability*, the swarm's ability to maintain or gain performance as robot numbers increase; and *partition-tolerance*, the ability of the system to operate even when connectivity is reduced due to lower swarm density.

Arena size The default arena sizes are $1\,\mathrm{m}^2$ in `ants2020` and $3.61\,\mathrm{m}^2$ in `scirob2023`. In experiments with constant swarm density, the area of the arenas is increased proportionally with the number of robots (as shown in Table 5.1). This is key to studying *scalability*, as it mitigates the adverse effects of overcrowded arenas and communication interference at large swarm sizes.

Type of Byzantine behavior and floor layout By default, Byzantine robots send estimates of 0 %, and the floors have the fixed layouts shown in Figure 5.1. In the *generality* study, the following changes are made:

- For each trial, a new floor layout is generated at random (keeping a constant black/white tiles ratio).
- Byzantine robots sample their estimates from a Bernoulli distribution with $p = 0.5$, yielding either 0 % or 100 % white tiles with equal probability.
- Byzantine robots sample their estimates from a uniform distribution, yielding values between 0 % and 100 %.

Time In the *resilience* and *hardware scalability* studies, we run the system continuously for 500 minutes to examine how hardware demands and token balances evolve over time.

(C) Evaluation metrics

For evaluating robustness, scalability and generality (Sections 5.3.1 to 5.3.3)

- **Consensus error** measures the difference between the actual percentage of white tiles in the environment and the smart contract estimate at the moment `hasConverged()` returns `true`. Lower values indicate that the swarm's estimate is closer to the true value.
- **Convergence time** represents the time (in minutes) until the convergence signal triggers; it is recorded when all robots have received the block where `hasConverged()` returns `true`. Lower values indicate faster convergence.
- **Reports per robot** represents the average number of transactions sent by a single robot before convergence. Lower values are better, as they indicate lower communication costs.

For evaluating resilience (Section 5.3.4)

- **Inbound token flow** is the cumulative amount of crypto tokens (including UBI and reward payments) received by a robot during a trial. Non-Byzantine robots are expected to accumulate more tokens and therefore have a higher inbound token flow compared to Byzantines.

For evaluating blockchain scalability (Section 5.3.5)

- **Average connected peers (ACP)** quantifies how well a robot remains connected to its peers. It is defined as the product of:
  - the fraction of the total experimental trial duration during which the robot had at least one peer, and
  - the average number of peers the robot had while connected to at least one peer.

  For example, a robot that was either connected to a single peer for the entire trial, or to two peers for half of the trial, has ACP = 1.

- **Average block period (ABP)** is the average of the observed intervals between consecutive blocks in the final version of the blockchain (not considering forks). Ideally ABP = B_p; however, delays are expected due to sealers which become unavailable due to network partitioning.

- **Blocks processed per minute (BPM)** is a measure of the processing load on the robots, corresponding to the number of blocks each robot receives per minute, averaged over each trial. Ideally BPM = 4, given that $B_p = 15$ s.

- **Block propagation time (BPT)** is the difference between the timestamps of a block's production and its reception by each robot, averaged over each trial. Ideally, BPT $< B_p$; otherwise, blocks are produced faster than they are disseminated among the robots.

- **Block production efficiency (BPE)** is the ratio between the blockchain length at the end of the trial and the total number of blocks produced by the robots. Higher values are desirable, as they indicate fewer wasted resources on blocks that are ultimately discarded.

For evaluating hardware scalability (Section 5.3.6)

- **CPU utilization** is the percentage utilization of each robot's onboard processor throughout the experiment.

- **Blockchain size** is the size (in MB) of the Ethereum `chaindata` directory where blocks are stored.

- **Cumulative traffic** is the total volume of data transmitted by a robot (in MB) up to a given moment during an experimental trial.

Table 5.2: Evaluation dimensions, metrics, and independent variables.

Dimension	Metrics	Independent variables
Robustness	Consensus error Convergence time Reports per robot	Number of Byzantine robots
Scalability – Constant arena size – Constant swarm density	Consensus error Convergence time Reports per robot	Number of robots
Generality	Consensus error Convergence time Reports per robot	Number of Byzantine robots Byzantine behavior Floor layout
Resilience	Inbound token flow	Time
Blockchain scalability – Constant arena size – Constant swarm density	ACP, BPM BPT, ABP, BPE	Number of robots
Hardware scalability	CPU utilization Blockchain size Transactions sent Cumulative traffic	Number of robots Time

5.3 Results

We evaluate the system across the six dimensions shown in Table 5.2. We first study how the smart contract secures swarm robotics consensus achievement and discuss the systems' *robustness*, *scalability*, and *generality*. Then we study *resilience*, analyzing how tokens are redistributed among Byzantine and regular robots and how this mitigates the impact of Byzantine robots. Finally, we assess the *scalability* of the blockchain protocol and of the hardware requirements.

5.3.1 Robustness

Summary Across datasets, the system withstands a moderate fraction of Byzantine robots (30–40%), while maintaining low consensus errors. However, as Byzantine numbers increase, convergence tends to require more time and reports. The `ants2020` dataset demonstrates that beyond a critical threshold, the Byzantine fraction dominates consensus, leading to biased outcomes. The `scirob2023` dataset demonstrates higher tolerance but also higher costs, potentially due to the stricter convergence criterion and the larger swarm size. Figure 5.2 shows these results.

Consensus error The `ants2020` dataset shows a median consensus error below 5 % for up to 3 Byzantine robots (30 % of the swarm). In `scirob2023`, errors remain under 2.2 % even with 9 Byzantines (37.5 % of 24 robots). Lower errors in `scirob2023` can be attributed to (i) a larger swarm (24 vs. 10 robots), which improves estimate quality through averaging; and (ii) a stricter convergence criterion (0.2 % vs. 1 %) Beyond a 40 % Byzantine fraction (`ants2020` only), errors increase sharply as the estimate oscillates between the true value and zero. At 50 %, variability decreases because Byzantines more consistently steer the value towards zero. This effect is absent in `scirob2023` due to lower Byzantine fractions.

Convergence time In the `ants2020` dataset, the average convergence time remains fairly stable, although variability tends to increase. In contrast, the `scirob2023` data shows a clear trend: convergence time also increases with the number of Byzantine robots. This is likely caused by the stricter convergence criterion, which is harder to achieve when fewer robots are sending good estimates.

Reports per robot Since the reports sent by Byzantine robots are typically rejected, robots need to send more reports to achieve convergence. This is particularly noticeable in `scirob2023` data, where the number of reports per robot increases with the proportion of Byzantine robots.

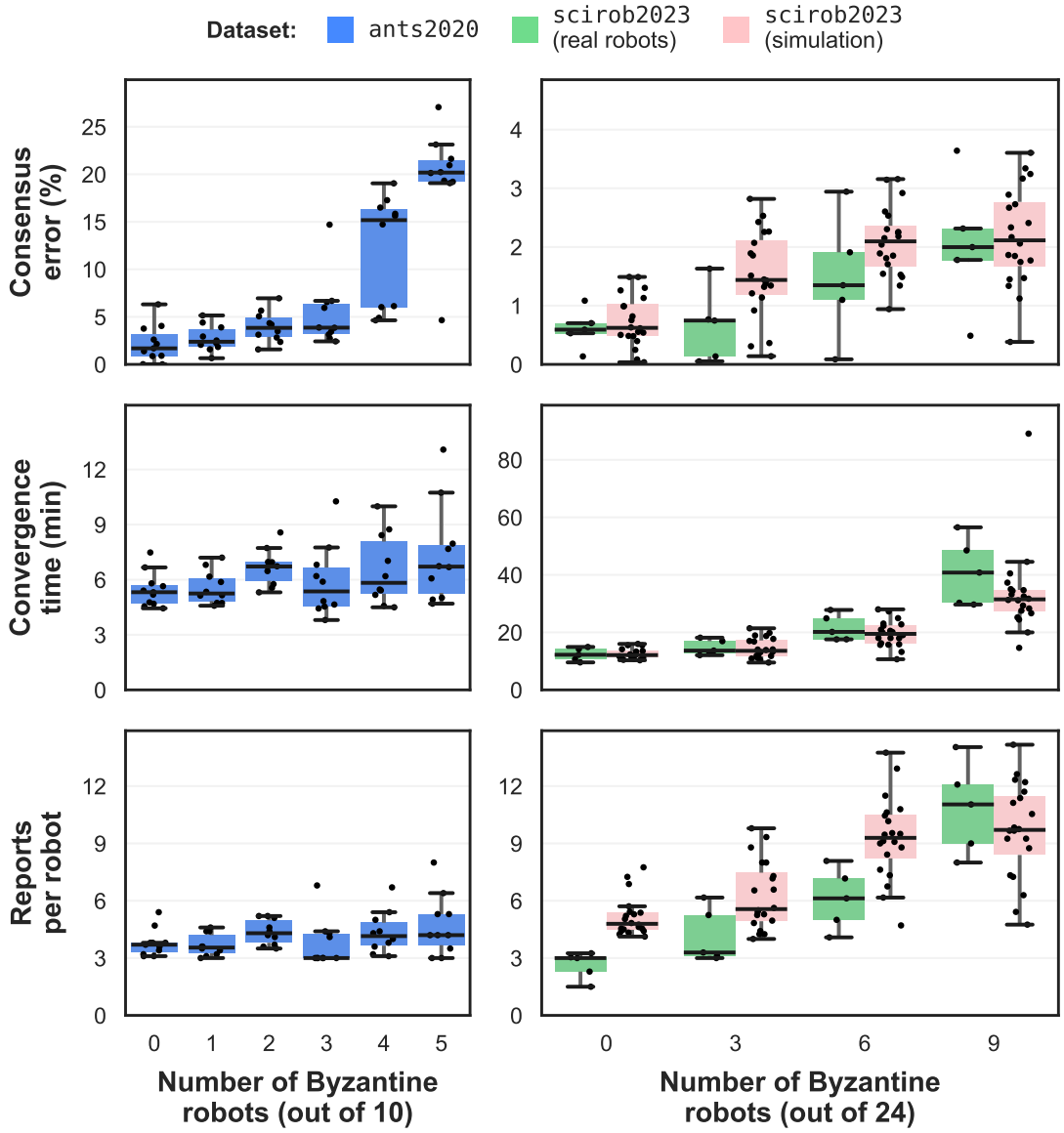


Figure 5.2: Robustness to Byzantine robots. Datasets: `ants2020` (10 trials, N=10 real robots) and `scirob2023` (5 trials on N=24 real robots and 20 trials in simulation). In `ants2020`, the Byzantine count increases from 0 to 5 (10 % steps, up to 50 % of the swarm). In `scirob2023`, it increases from 0 to 9 (12.5 % steps, up to 37.5 %). The Byzantine robots always report 0 % to the smart contract.

5.3.2 Scalability

Scalability results are divided into two settings: (A) Constant arena size, where we increase the number of robots in a fixed-size arena; and (B) Constant swarm density, where we increase arena dimensions proportionally with the number of robots. Swarm size affects performance differently in each setting. Under constant arena sizes, it primarily accelerates convergence, while under constant swarm density, it preserves low consensus errors with only moderate increases in convergence time. In general, low consensus errors across all arena and swarm sizes, combined with mild increases in convergence time and reports, support the *scalability* of a blockchain-based approach for secure consensus achievement in swarm robotics.

(A) Constant arena size

Summary The consensus error remains low, indicating that accuracy does not degrade as more robots are added. Convergence time consistently decreases with increasing swarm size, showing that additional robots speed up consensus. At the same time, the number of reports per robot remains roughly constant, keeping per-robot communication costs stable regardless of swarm size. These results are shown in Figure 5.3.

Consensus error The median error remains low regardless of swarm size, and the "wisdom of the crowd" effect (Galton 1907) is not very pronounced, as the median error does not decrease with increasing swarm size. This may be due to two reasons: (i) larger swarms converge faster, resulting in shorter trials; and (ii) the robots are homogeneous, exhibiting similar sensor quality. The `ants2020` dataset does show a pattern where the median error increases with swarm size (up to 10 robots at least), which may be due to noise caused by the looser convergence threshold.

Convergence time Convergence time decreases consistently with swarm size, showcasing that when the arena size does not increase, the swarm effectively distributes the workload (in this case, the task of sensing the fixed-size arena floor) across the many robots to achieve faster convergence.

Reports per robot The number of reports per robot remains roughly constant, indicating good scalability of the simple gossip-based protocol, since the number of messages increases only linearly with the number of robots.

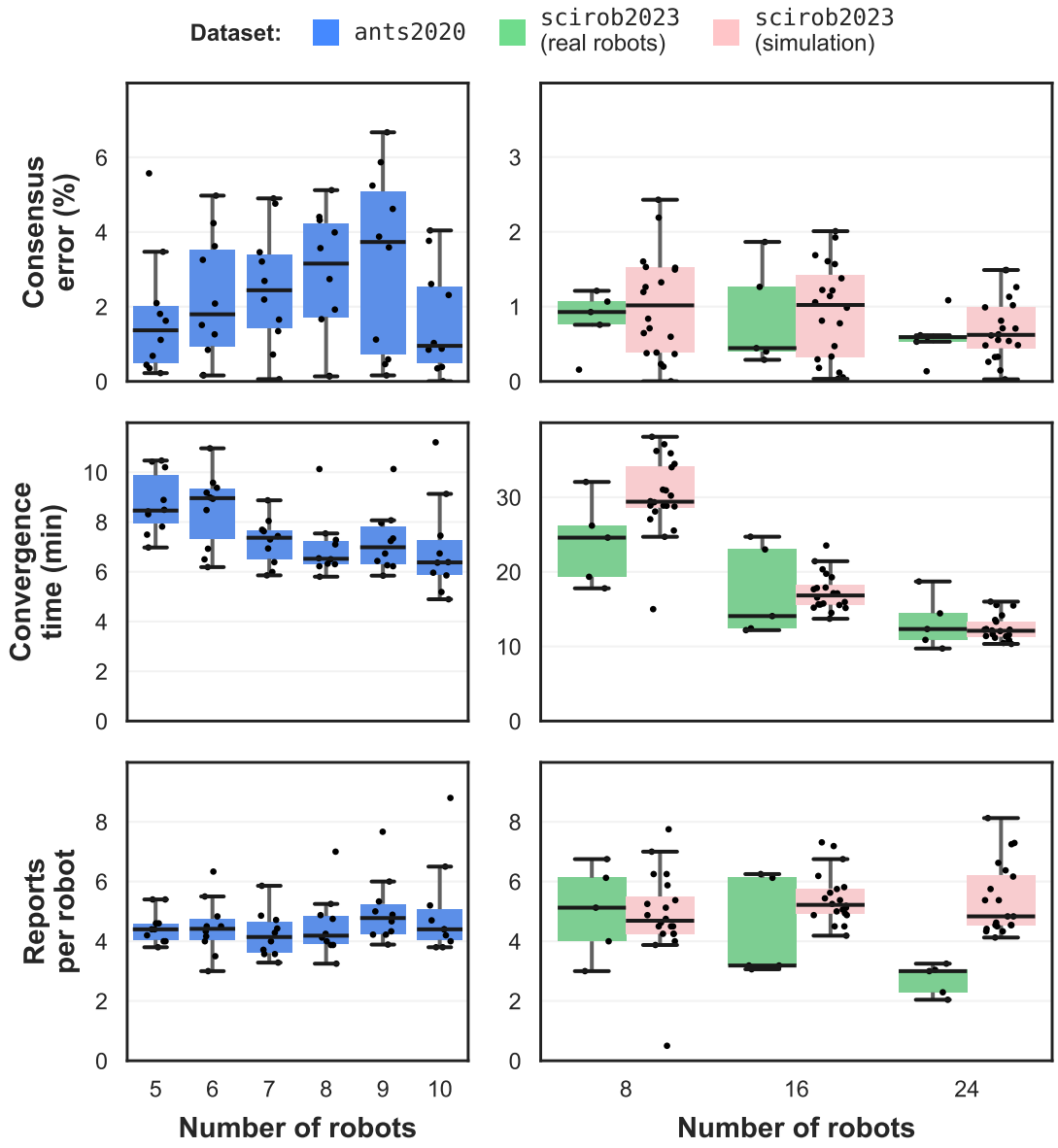


Figure 5.3: Scalability with constant arena size. Datasets: `ants2020` (10 trials on real robots) and `scirob2023` (5 on real robots; 20 in simulation), without Byzantine robots. In `ants2020`, we use a 1 m^2 arena, adding one robot at a time to increase density from 5 to 10 robots per m^2. In `scirob2023`, we use a 3.61 m^2 arena with swarms of 8, 16, and 24 robots, thereby increasing density from 2.2 to 6.65 robots per m^2

(B) Constant swarm density

Summary The consensus error remains low and improves slightly as swarm size increases, with both variance and median error decreasing. Convergence time grows modestly in comparison with the number of robots, while the number of reports per robot remains approximately constant, confirming excellent scalability of the gossip-based protocol even in large swarms. Results are shown in Figure 5.4.

Consensus error The median consensus error remains below 3 %, both in simulation and reality. Both the variance and the median error decrease as the number of robots increases, indicating that accuracy improves with the number of robots, since noise is averaged out as the number of robots increases in larger arenas.

Convergence time The minutes to achieve convergence grow only modestly with swarm size: from 8 to 120 robots (a 15× increase), the median time to consensus increases by less than a factor of two. This increase likely reflects the longer disconnection periods that may occur in large arenas, since some robots remain isolated in arena corners, leading to longer delays in propagating the consensus achievement signal.

Reports per robot The median number of reports per robot required to reach consensus is approximately constant, with variability decreasing as swarm size grows (due to averaging over more robots).

5.3.3 Generality

To assess whether the methods deployed remain flexible when experimental conditions change, we reassess the system's robustness and performance when changing key experimental conditions. As shown in Figure 5.5, the performance under different conditions is similar to that observed under the default conditions of the previous result sections. This indicates that the proposed methods are flexible to changes in the tile distribution as well as to a spectrum of Byzantine behaviors. However, as a proof of concept, this work did not intend to address general robustness or applicability to different consensus achievement problems. In fact, certain Byzantine behaviors can potentially break the system, and it is not clear how many Byzantine robots the system can withstand. These limitations are further discussed in the next section, and in Chapter 6 we present a protocol that withstands arbitrary Byzantine behaviors.

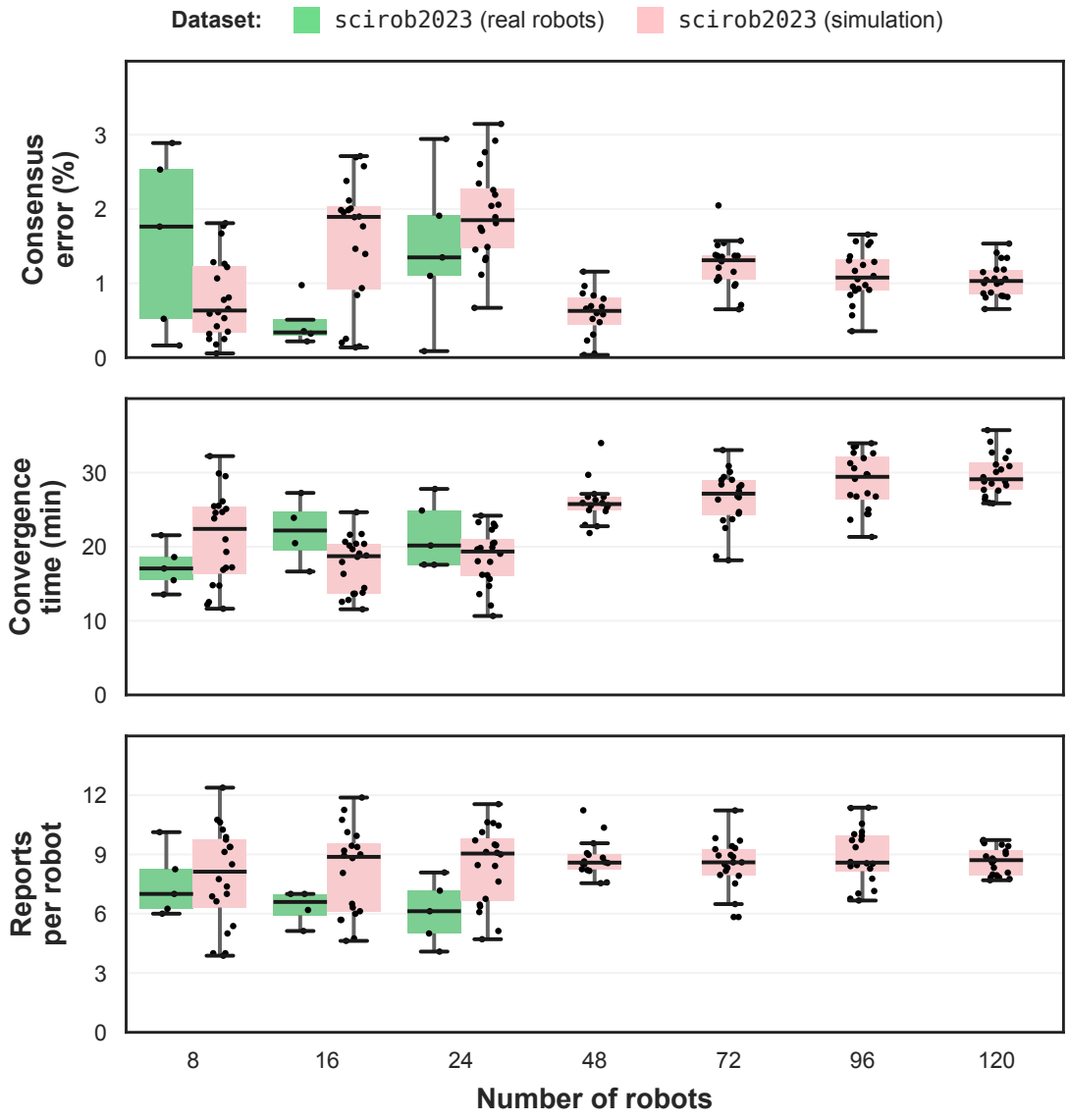


Figure 5.4: Scalability with constant swarm density. Datasets: `scirob2023` on real robots (5 trials per swarm size) and in simulation (20 trials per swarm size), with 25% Byzantine robots. Arena area scales proportionally with swarm size to maintain constant density of 6.6 robots per m^2. For swarm sizes above 24 (simulation only), the number of robots increases in steps of 24 rather than 8. The `ants2020` dataset is unavailable as that study comprised a single arena size.

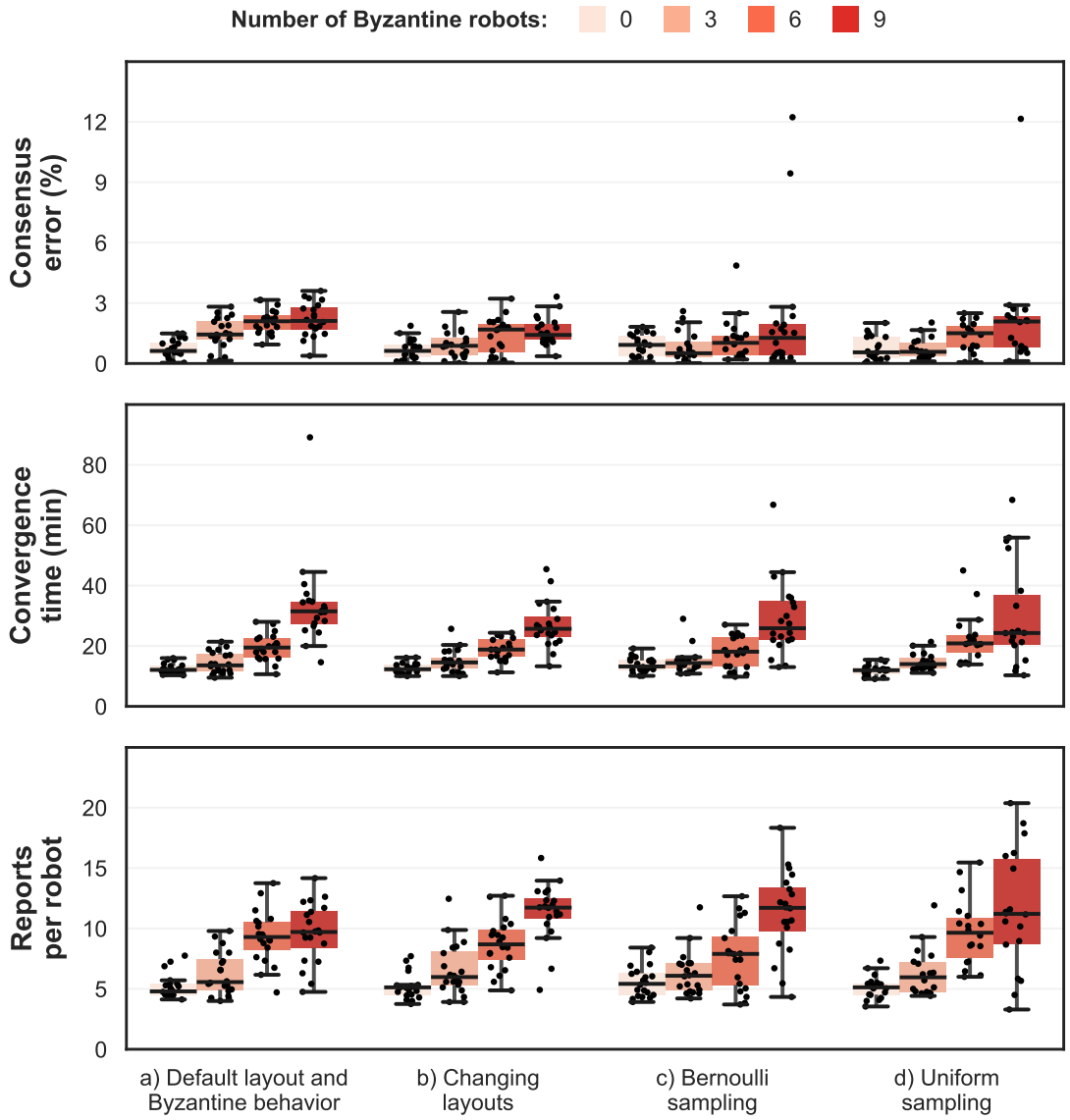


Figure 5.5: Generality study with increasing numbers of Byzantine robots. Dataset: `scirob2023` on 24 simulated robots (20 trials per configuration) in a 3.61 m^2 arena. a) Baseline results from Section 5.3.1. b) For each trial, a new random floor is generated with constant black and white tile ratio, while Byzantine robots report a constant 0 % estimate. c) Byzantine robots sample their estimates from a Bernoulli distribution estimate ($p = 0.5$, yielding 0 % or 100 %). d) Byzantine robots sample from a uniform distribution.

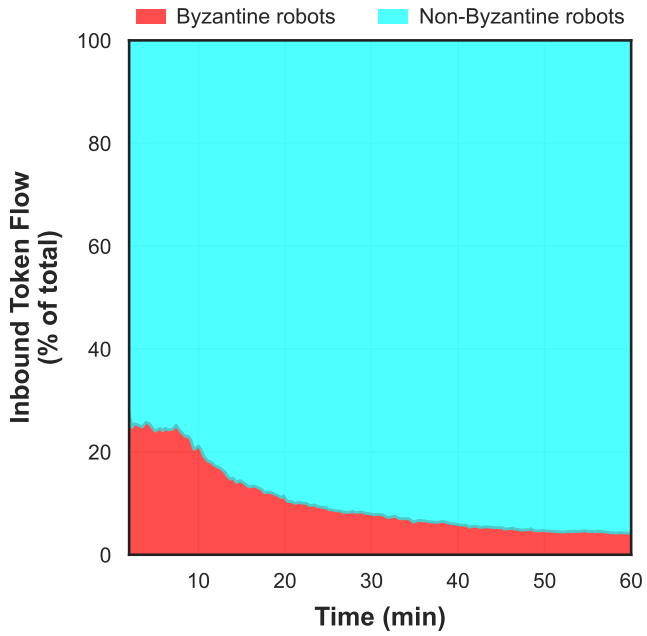


Figure 5.6: Inbound token flow over time. Dataset: `scirob2023` on simulated robots (20 trials lasting 1 h each). The swarm density remains constant across sizes, and there are 25% Byzantine robots present in the experiment. The plot shows the fractions of the inbound token flow received over time by the non-Byzantine robots (cyan area) and Byzantine robots (red area).

5.3.4 Resilience

Our system mitigates the impact of Byzantine robots over time through a crypto-token economy that regulates how frequently robots may submit estimates for consensus. A robot's inbound token flow can come from two sources: (i) universal basic income (UBI) payments, or (ii) rewards issued by the smart contract. Robots with higher inbound token flow can submit estimates more frequently and therefore exert greater influence on consensus achievement. As shown in Figure 5.6, the flows of Byzantine and non-Byzantine robots diverge over time. Initially, all robots receive UBI payments; since Byzantines constitute 25 % of the swarm, they receive 25 % of the total inbound token flow at the start. However, as UBI payments become increasingly sparse, rewards from accepted estimates become the primary source of income. Non-Byzantine robots capture most of these rewards as Byzantine robots' reports are rejected by the outlier detection mechanism. As a result, Byzantine robots progressively lose the ability to submit estimates and their influence on consensus diminishes significantly.

5.3.5 Blockchain scalability

We now analyze the scalability of the blockchain consensus and synchronization processes. Our results reveal the scalability limits of proof-of-authority under increasingly large, mobile swarms, as efficiency degrades with increasing swarm sizes. However, it appears to stabilize at very large swarm sizes, likely because of implicit averaging of delays across the network which mitigates variability.

(A) Constant arena size

Summary The increased connectivity (APC) in larger swarms reduces block propagation time (BPT). In `ants2020`, however, the network reaches a saturation point around 9–10 robots. At this point, blocks propagate more slowly and the block production efficiency (BPE) drops sharply, resulting in more blocks processed per minute (BPM) and thereby wasted computational and communication resources. Figure 5.7 shows these results.

Average connected peers (ACP) Smaller swarms exhibit lower ACP, meaning that robots are less connected. In both datasets, we observe diminishing returns as swarm size grows; and `ants2020` shows a converging trend as communication signal interference increases with swarm size in the small $1\,\mathrm{m}^2$ arena.

Blocks per minute (BPM) Ideally, the average BPM is 4 blocks, given the 15 s block period. In practice, isolated robots occasionally received up to 15 blocks from a single peer at once (not visible in the plot due to averaging over trials). In both datasets, BPM *increases* with swarm size; the main reason being that more blocks are produced, exchanged and discarded (see BPE). In `scirob2023`, however, the sparser network (lower robot density in the arena) leads to delays in block production (see ABP), driving BPM downward relative to `ants2020`.

Block propagation time (BPT) The time it takes for blocks to be propagated in the network generally tends to decrease with connectivity, except in `ants2020` at 9–10 robots, where saturation and communication interference slow propagation.

Average block period (ABP) Due to network conditions, ABP deviates from the preset 15 s. As the swarm size increases, however, ABP tends to converge towards this value, indicating that more connectivity between robots reduces the period between blocks. With 8 robots in the $3.61\,\mathrm{m}^2$ arena, we observe the highest block periods, about 30 seconds.

Block production efficiency (BPE) The BPE decreases with swarm size: in `scirob2023` with 24 robots, fewer than 25 % of produced blocks end up as part of the blockchain due to sparse communication. Despite better connectivity with higher swarm sizes, there are more robots producing blocks out of turn when the reception of blocks from preferred sealers is delayed.

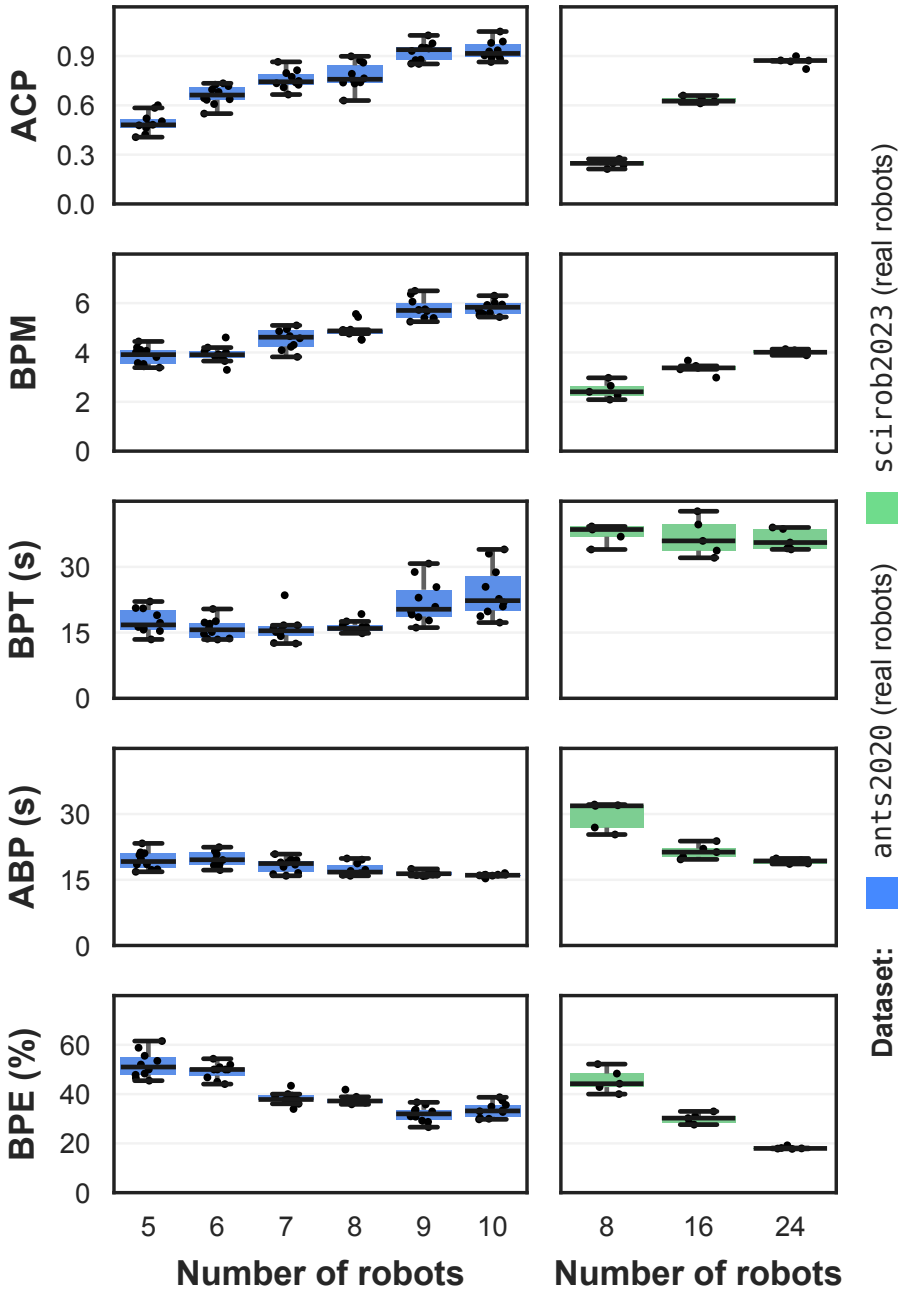


Figure 5.7: Blockchain scalability with constant arena size. Datasets: `ants2020` and `scirob2023` on real robots (10 trials on a 1 m^2 arena, and 5 trials on a 3.61 m^2 arena, respectively), without Byzantine robots.

(B) Constant swarm density

Summary BPT grows with swarm size which means that, on average, blocks take longer to arrive to robots. Correspondingly, the BPE falls below 20% (i.e., four out of five produced blocks are discarded). This seems to indicate that the preset 15 s block period is too short for large swarms. Notably, BPT tends to stabilize, suggesting that efficient operation at very large scales might be possible if a sufficiently large block period is chosen. These results are shown in Figure 5.8.

Block propagation time (BPT) The BPT increases with swarm size. At large scales, blocks take on average about 40–60 s to reach other robots, while a new block is produced every 15 s. Because propagation lags behind production, proof-of-authority sealers create blocks out-of-turn when they do not receive the latest blocks from the preferred sealers. Raising the block period B_p above the observed BPT would improve BPE. However, a larger B_p slows smart-contract execution and reduces system responsiveness.[1]

Average block period (ABP) The ABP tends to increase with swarm size. In proof-of-authority, a sealer cannot produce more than one block within the most recent $\left\lfloor \frac{N}{2} \right\rfloor$ blocks. If a newly produced block does not reach an eligible sealer within B_p seconds, the next block may be delayed, causing the effective block period to exceed the nominal 15 s. There seems to be a large simulation gap, particularly with 8 robots. This occurs because with real robots, it is difficult to enforce the exact communication range, as it is regulated through the power of the infrared transmission which, due to imperfect power control and environment reflections, can still achieve sporadic communication links at a higher range. Therefore, real robot networks are better connected than the simulated ones, as evidenced in the figure.

Block production efficiency (BPE) The BPE decreases steadily as the swarm size increases. For swarms larger than 48 robots, median BPE drops below 20 %, meaning that, on average, 4 every 5 produced blocks end up being discarded, leading to wasted computational resources and bandwidth. Although blockchain convergence is still achieved, this reveals scalability limitations in proof-of-authority. In large networks, the preferred sealers may fail to produce and disseminate their blocks in a timely manner, prompting other robots to produce their own concurrent blocks. The resulting blockchain forks persist until one branch gains sufficient adoption (particularly by preferred sealers).

[1] Reducing the block period to improve real-time coordination is implemented and discussed in Chapter 7 – Section 7.2.

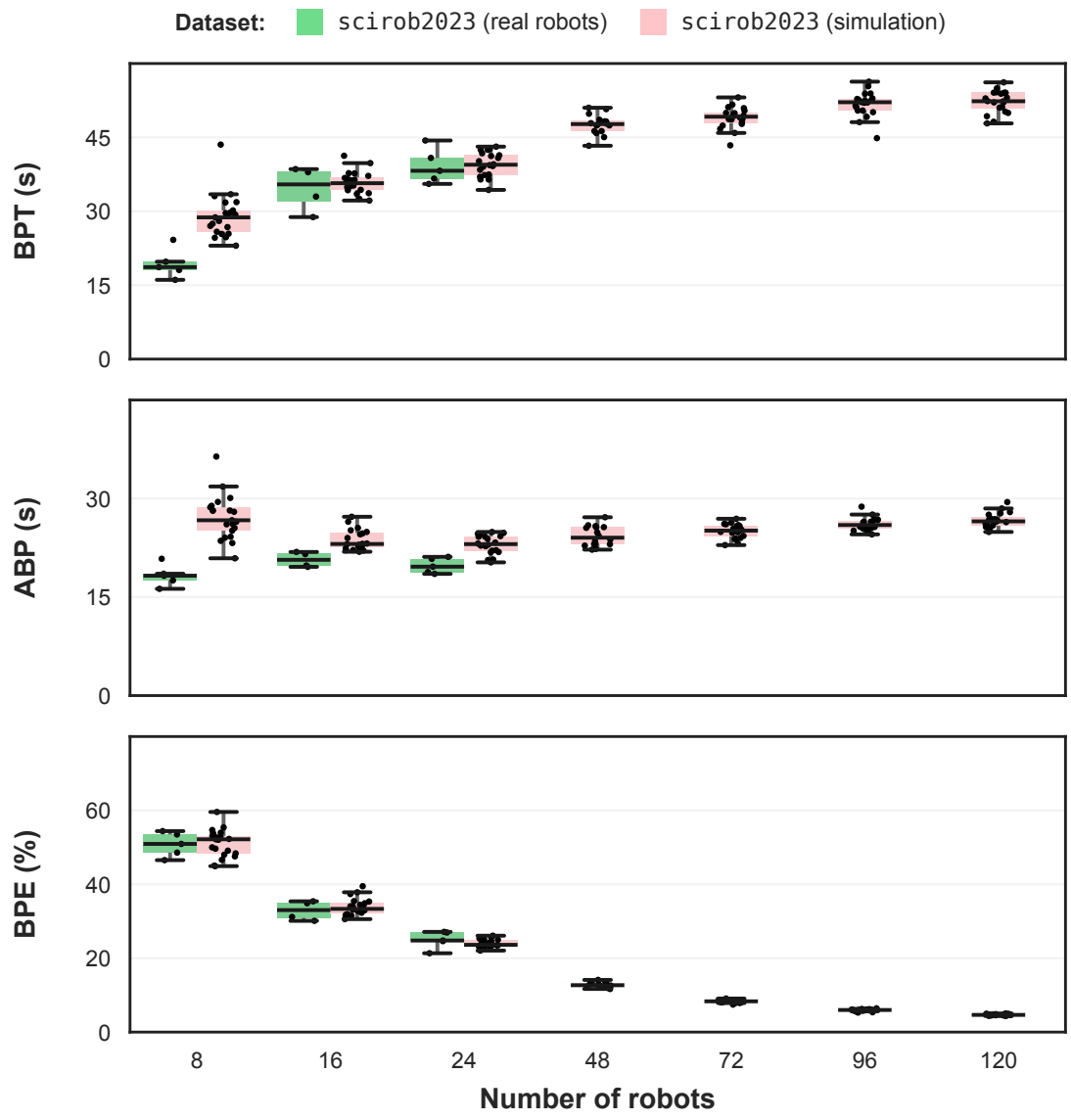


Figure 5.8: Blockchain scalability with constant swarm density. Datasets: `scirob2023` on real robots (5 trials each swarm size) and in simulation (20 trials), with 25% Byzantine robots. For swarm sizes above 24 (simulation only), the number of robots increases in steps of 24 instead of 8.

5.3.6 Hardware scalability

Our results show that the system scales without introducing hardware bottlenecks. Processing demands remain stable as the number of robots increases, while both data storage and network traffic grow linearly with the experiment duration and the swarm size. This behavior follows directly from the peer-to-peer communication protocol and the append-only nature of the blockchain. Overall, these results indicate that the proposed method is suitable for long-term operation and large-scale deployments under realistic conditions.

Processing Figure 5.9 reports CPU utilization on the Raspberry Pi Zero W, averaged over each trial and across all robots. In `ants2020`, CPU utilization increases as the constant-size arena becomes increasingly crowded, reaching densities of up to 10 robots per m^2. In `scirob2023`, this trend is less pronounced, as connection events occur more sparsely in the larger arenas. Overall, with constant arena size, CPU load increases with swarm size, whereas this is not the case with constant density.

Storage To evaluate long-term storage requirements, we ran a single simulation for 500 minutes. As shown in Figure 5.10 (top), the blockchain size grows linearly with both time and the number of robots, resulting in predictable storage requirements. Given that the Pi-pucks are equipped with 16 GB SD cards, the same experiment could be run continuously for several weeks before exhausting available storage. Storage capacity is therefore not a limiting factor in our system. For the swarm size of 120 robots, a spike in storage usage occurs at approximately 450 minutes. This is caused by a scheduled LevelDB compaction triggered by the Ethereum software.

Communication To compute communication overhead, we monitored the packets received by the robots. As shown in Figure 5.10 (bottom), each robot transmitted between 10 MB and 72.5 MB of data in total, depending on the swarm size. For a given swarm size, per-robot variability arises because better-connected robots relay more traffic. This effect is visible in the shaded bands in the left plot and the violin plot distributions on the right. Overall, the total volume of transmitted data is approximately equal to the final blockchain size, but not exactly the same because (i) the reported blockchain size includes auxiliary files, (ii) packet loss, and (iii) block headers associated with forks that are not part of the final chain. Our results indicate that communication overhead is minimal: each robot transmit and receives data approximately equal to the final blockchain size, and bandwidth utilization remains stable over time. Consequently, the Pi-puck's onboard Wi-Fi module operates at less than 1 % of its available bandwidth.

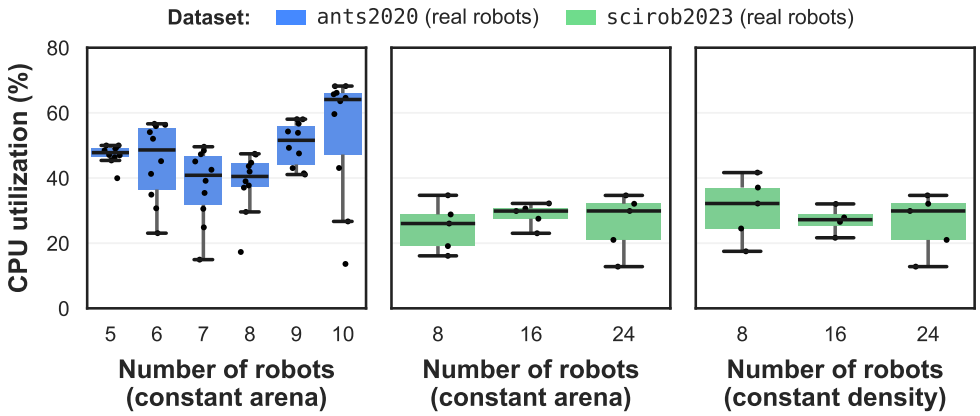


Figure 5.9: Processing costs as swarm size increases. Datasets: `ants2020` and `scirob2023` (10 and 5 trials on real robots, respectively). In `ants2020`, swarm size increases from 5 to 10 robots, one robot at a time, while `scirob2023` includes 8, 16, and 24 robots. Results are shown for both constant arena size and constant swarm density experiments.

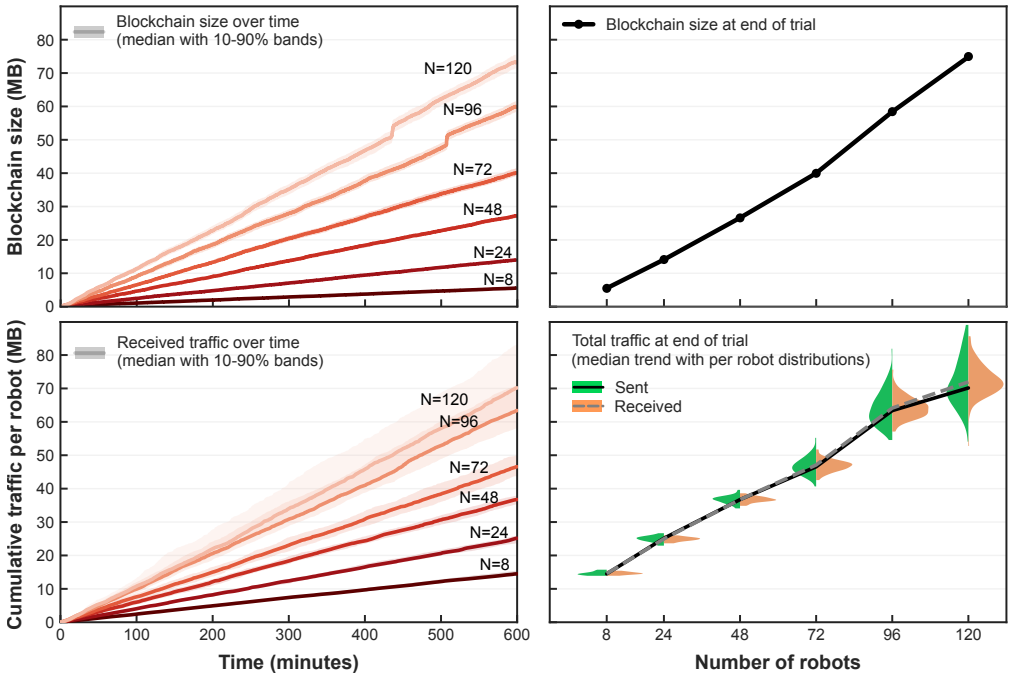


Figure 5.10: Blockchain storage and communication costs as swarm size increases. Dataset: `scirob2023` on simulated robots (one trial per swarm size with a duration of 500 minutes, constant swarm density, and 25% Byzantine robots).

5.4 Discussion

Our experimental results demonstrate the practical feasibility of using a blockchain to secure consensus achievement in a robot swarm. They also show that the *operational costs of running a blockchain are manageable for robot swarms.* Communication and storage demands scale linearly with swarm size, and further efficiency gains could be achieved by optimizing and exploiting blockchain data structures.

On the downside, the consensus protocol used—while secure and functional—is not optimized for the highly dynamic topologies of robot swarms, leading to severe communication inefficiencies. Indeed, the block production efficiency (BPE) can fall significantly at larger swarm sizes (down to 5–20%). This increases the number of synchronization exchanges required to maintain a consistent blockchain and leads to wasted computations when robots process blocks that are eventually discarded. However, we also observed that the total volume of data exchanged does not meaningfully exceed the data stored on-chain, since the data-heavy transactions are only downloaded once by each robot, and remain available even when blocks are discarded. Nonetheless, Wi-Fi interference arises not only from message size but also from the frequency of exchanges, presenting a scalability challenge that limited the experiments to 24 robot swarms. Two technical strategies can mitigate this issue:

- **Increasing the block period:** Allowing more time for block propagation reduces forks and improves BPE.
- **Reducing Wi-Fi traffic:** Moving metadata exchanges (block headers, identities, transaction roots) to scalable communication channels such as infrared or NFC radio frees Wi-Fi bandwidth for large data transfers, potentially enabling deployments with larger and denser swarms.

5.4.1 Vulnerabilities and mitigation

In the following, we outline how more complex Byzantine robots (e.g., robots controlled by an attacker) could exploit the system and how the design bounds the impact of these attacks, noting stronger mitigations and their trade-offs where relevant.

Eavesdropping Exploiting the fact that transaction data is public, Byzantine robots could copy peers' estimates to unfairly gain tokens or bias consensus (e.g., by submitting values that barely qualify as inliers). This vulnerability stems from blockchain transparency: all nodes must access the same data to independently construct a consistent state. In our design, the potential damage is bounded: the contract accepts only estimates within ± 0.1 of the current swarm estimate,

and we assume honest estimates dominate because Byzantines are a minority. A stronger defense involves a commit-reveal scheme, where robots first submit a cryptographic commitment of their estimate and disclose the value only after enough commitments are collected. The smart contract verifies each commitment consistency and penalizes invalid reveals. This prevents eavesdropping but it requires an additional communication round, slowing consensus speed.

Delayed Byzantine behaviors Byzantine robots can initially adhere to the rules before switching to Byzantine behavior once they have accumulated a reasonable amount of tokens. In our design, the impact is limited because token acquisition is gradual: robots earn tokens via UBI (which decays exponentially over time) or via rewards for accepted estimates. Thus, accumulating large wealth relative to honest peers is difficult unless the Byzantine robot submits accepted estimates over long periods while others incur losses from occasional faults (e.g., sensor noise or environmental interference).

Majority attacks An adversary controlling a majority of the robots could manipulate the blockchain's proof-of-authority consensus, thereby compromising the smart contract and token economy. While proof-of-authority can typically withstand 50% malicious participants, under non-ideal conditions (out-of-sync robot clocks and delayed communications), disruption may be possible with one-third of the robots (Ekparinya et al. 2020). Our approach therefore relies on the security of the underlying consensus. Still, assuming an honest majority is reasonable: the smart contract defenses would not resist such a large Byzantine fraction in any case.

5.5 Challenges and future work

This section summarizes the main limitations of our current approach and outlines promising directions to extend it toward more realistic deployments, including stronger adversaries, network partitions, churn, open membership, and time-delay constraints.

5.5.1 Complex tasks and stronger adversaries

At this stage the objectives were foundational: bringing blockchain to physical robots to demonstrate feasibility and evaluate key properties. The black-and-white tile scenario is common for proof-of-concept studies in swarm robotics literature, and the smart contract, though simple, illustrates the potential of our approach. A natural next step is to position the work as a general framework for secure consensus in robotics, which would require implementing a generic smart contract applicable across consensus tasks and applications. This thesis provides further

advances on this.

In Chapter 6, a smart contract is presented for generic consensus achievement tasks in robot swarms, supporting both discrete and continuous agreements. It is deployed for consensus on resource coordinates (Zhao et al. 2023†) (simulation) and on RGB values from robot cameras (Pacheco et al. 2025†) (simulation and physical experiments). Chapter 7 also introduces application-specific contracts, including federated learning (discarding Byzantine model weights) (Pacheco et al. 2024a†) and Swarm-SLAM (using geometric constraints to detect map inconsistencies and penalize Byzantines) (Moroncelli et al. 2024†).

5.5.2 Network partitions and collusion

Real deployments may experience prolonged partitions that delay information propagation and generate disconnected sub-swarms. Adversaries can potentially *join forces to manipulate sub-swarms (collusion).* In this study, a Byzantine collusion attack could synchronize their transactions to gain a majority in the following consensus round. To strengthen their attack, they could even target an isolated block producer which would not receive as many transactions from honest robots. One mitigation strategy could be to avoid network partitions through self-organized aggregation strategies (as opposed to random walks). While this would mitigate the problem, it would not resolve it if, for example, the network partitioned due to a physical event such as a falling tree or malicious communication jamming.

In Chapter 6, we address this issue through a collusion-resistant protocol implemented through the smart contract, showing how it fares against attacks performed by Byzantine collusions of up to one-third of the swarm. This protocol is capable of securing consensus even when severe network partitions occur.

5.5.3 Inactive robots

Over long operations, it is inevitable that robots break down. Each time an otherwise non-Byzantine robot stops functioning, Byzantine robots become increasingly dominant since they contribute a larger portion of the estimates. Eventually, the system can become compromised unless a defense mechanism is in place.

Chapter 6's protocol addresses this problem by generating new tokens which are rewarded to robots that actively send their estimates, indirectly penalizing robots that are idle or disabled. Over long runtimes, this constitutes an autonomous mechanism for fault recovery that prevents Byzantine robots from gradually gaining power as other robots fail.

5.5.4 Dynamic and open swarms

Allowing robots to join/leave or to be owned by different parties raises issues such as Sybil attacks (e.g., a malicious robot joins using different identities to collect UBI). In a single-stakeholder and small-scale deployment, access is controlled by a system administrator that adds or removes robots from the swarm. However, this solution may not apply to deployments of large swarms that operate autonomously, or when multiple competing stakeholders own the robots.

Existing permissionless blockchains require the spending of computational resources (such as energy or storage space), which is not feasible in mobile robotics as robots would need to compete with non-mobile hardware. One possibility is to develop self-governance for robot swarms, establishing the rules by which membership or roles can be changed. External stakeholders can vouch for their robots through staked tokens, thereby being accountable for robots' actions. In this way, security is contingent on the joint vested interest of the stakeholders.

5.5.5 Coordination delays

The use of blockchain technology introduces time delays, since state updates are only executed when new blocks are created and propagated. In this case, we used the default value of 15 seconds, and showed that in large swarms this is not sufficient, since blocks can take up to 60 seconds to be received by peers. Setting a lower block time is therefore a trade-off: lower block times may be required to coordinate or supervise robots in real time (Pacheco et al. 2022[†]), but they also increase the probability of blockchain forks and conflicts.

In this chapter, the delay introduced by the blockchain was not problematic because the smart contract did not handle low-level control or swarm organization. In Chapter 7 – Section 7.2, we explore a foraging scenario where the smart contract directly provides action plans to robots. In this case, long block update times can negatively impact performance, so the block period is reduced to 2 seconds (Pacheco et al. 2022[†]). Other potential solutions include replacing the blockchain with a different distributed ledger technology offering higher throughput or improved partition tolerance, such as a DAG-based architecture (Danezis et al. 2022), or developing consensus protocols tailored to robot swarm network conditions.

5.6 Chapter summary

In this chapter, we demonstrated how blockchain technology can protect a robot swarm from Byzantine robots that spread incorrect information. Using a smart contract, the system aggregates consensus reports from many robots to produce a

swarm estimate more robust than traditional methods. Leveraging the blockchain's resistance to double-spending, we introduced a token economy that regulates robot participation in consensus tasks. Robots that submit erroneous information lose tokens and are eventually excluded from consensus.

We also demonstrated the feasibility and practicality of using blockchains to secure and potentially coordinate robot swarms. Scalability was a key concern when this work was first presented (Pacheco et al. 2020[†]). By extending our setup to 24 real robots and 120 simulated robots, and running experiments lasting up to 10 hours, we showed that hardware requirements remain reasonable, either constant or linear in swarm size. Although blockchain synchronization remains possible at large scales and in sparsely connected swarms, its efficiency decreases. Future work on dedicated consensus protocols for swarm robotics is therefore required.

Chapter 6

Swarm Oracle: Reaching Trustless Global Agreements

Blockchain consensus, rooted in the principle "don't trust, verify", limits access to real-world data as it may be ambiguous, noisy or otherwise unverifiable by all participants. Oracles address this limitation by supplying data to blockchains, but existing oracle solutions often compromise autonomy and transparency, or reintroduce trust. This tension between blockchains' decentralization and access to real-world information is commonly referred to as the *oracle problem*.

In this chapter we address the oracle problem by interfacing swarm-robotics hardware with blockchain software. We introduce *Swarm Oracle*, a decentralized network of autonomous robots that use onboard sensing and peer-to-peer communication to collectively validate real-world information and deliver it to a blockchain.

Swarm Oracle serves two roles. First, it enables a robot swarm to securely update a blockchain-based collective memory with information that robots individually observe in the environment. This information provides a reliable substrate for swarm coordination and self-organization that is grounded in global knowledge produced through decentralized agreement rather than local information. Second, Swarm Oracle can act as a mobile sensor network that answers on-demand information requests from smart contracts on public blockchains. Here, swarm robotics' decentralization, mobility, and fault tolerance are key to addressing the oracle problem in situations where static sensors or centralized oracles could become inflexible solutions or potential points of failure.

In either role, it is crucial that Swarm Oracle provides correct and timely information; otherwise, decisions grounded in incorrect global knowledge could have serious consequences, and stalled updates could block decision-making and coordination processes. Swarm Oracle must therefore remain reliable both in the presence of faults and attacks from adversaries. To remove potential biases or

errors injected during the robots' manufacturing or programming, Swarm Oracle is envisaged to consist of robots from competing stakeholders, with potentially misaligned goals.

To achieve the required robustness, we implement a Byzantine fault-tolerant protocol as a smart contract. This smart contract can be deployed on either a public blockchain, or in a robot-hosted blockchain. In this work, we deploy the contract on a blockchain hosted by a swarm of 12 robots. The blockchain ensures (i) a globally consistent ordering and deterministic processing of robots' data; and (ii) an immutable ledger of reputation tokens. The reputation mechanism supports long-term resilience by reducing the influence of Byzantine robots and enabling the swarm to recover autonomously from faults and malicious attacks over time.

The remainder of this chapter is structured as follows. Section 6.1 motivates the oracle problem and positions Swarm Oracle with respect to existing decentralized oracle networks and swarm-robotics consensus, which were reviewed in Chapter 2. Section 6.2 introduces the collective perception environment and task, provides a short intuitive overview of Swarm Oracle's operation in the context of this experimental scenario, describes the robot controllers, the Byzantine faults and attacks, and the experimental setup. Section 6.3 discusses the results obtained using 12 physical robots and simulations, quantifying agreement accuracy, robustness, costs, and the long-term dynamics of the reputation-based resilience mechanism. Section 6.4 discusses scalability, vulnerabilities, mitigation strategies, and deployment considerations. Finally, Section 6.5 presents the Swarm Oracle's Byzantine fault-tolerant protocol, reputation mechanism, and smart contract implementation in detail.

6.1 Introduction

To operate at a global scale, blockchain systems are designed to be decentralized, transparent, and trustless, relying on large networks of nodes to independently verify that new blocks follow consensus rules and that user transactions are valid. However, this design imposes a fundamental limitation: blockchains cannot securely incorporate information unless it is algorithmically verifiable by every node. This limitation also applies to smart contracts that rely on accurate real-world data to support decision-making, such as exchange rates for decentralized finance, seismic activity measurements for disaster insurance, or environmental indicators for managing smart cities and natural resources (Antonopoulos and Wood 2018).

Entities that feed real-world information into a blockchain are known as oracles—a name borrowed from the oracles of ancient Greece, reputed to convey divine, all-seeing knowledge. Yet introducing an oracle also introduces a single point of trust and, often, centralization to the system: the blockchain's behavior now

depends on the integrity and availability of an external data source. This can undermine key blockchain properties, such as automated execution, censorship resistance, and transparency. This issue is commonly referred to as the blockchain "oracle problem" (Caldarelli 2020).

In Chapter 3 we provided an initial introduction to the oracle problem and how it limits blockchain applications—including those in robotics. In Chapter 2 we reviewed existing approaches to address or mitigate the oracle problem, in particular decentralized oracle networks (DONs) such as Chainlink (Breidenbach et al. 2021), and the limited instantiations of hardware oracles that collect data directly from the environment.

In the previous chapter, we showed how a blockchain lets robot swarms synchronize a global computation state and provides a secure substrate for the detection of inconsistent or extreme values reported by some robots; and how crypto tokens can be used to moderate robots' participation and incentivize good behaviors (Pacheco et al. 2020†, Strobel et al. 2023†). However, the "oracle agreement" in these studies—in which robots vote and agree on the fraction of white floor tiles—offered no formal guarantees on achieving a global consensus, and remained vulnerable to malicious behaviors such as those exploiting locality (as discussed in Section 5.4). A small group of coordinated robots could potentially synchronize their votes to exploit the system, thereby gaining crypto-tokens and voting power.

In this chapter, we deploy a Byzantine fault-tolerant consensus protocol on a swarm of 12 physical robots, to form an experimental Swarm Oracle. We stress-test the system at its theoretical limit of fault tolerance: when one third of the robots coordinate attacks to compromise the Swarm Oracle's *safety* and *liveness* properties[1] (Lamport 1977). We also integrate a reputation system that enables Swarm Oracle to self-heal from faults and attacks, as this is a crucial requirement for continuous and autonomous operations. Unlike previous reputation systems (Pacheco et al. 2020†, Strobel et al. 2023†, Zhao et al. 2023†) that penalized only robots providing unreliable information, Swarm Oracle also safeguards liveness by penalizing robots that become inactive, either due to accumulating faults or attacks. In our experimental scenario, robots use onboard cameras to identify the colors of features in the environment. This scenario reflects real-world conditions where robots have varying levels of accuracy and individual biases in their sensing, as color perception is highly influenced by environmental and hardware variability (e.g., camera calibration and lighting conditions). We perform an extensive experimental study using real robots (approximately 22 hours of runtime) and simulated robots (approximately 278 hours of runtime), showcasing how the Swarm Oracle provides

[1]The safety property states that *bad things won't happen*—in our case, that the robots will not agree on incorrect information. The liveness property states that *good things will happen*—in our case, that agreements are reached within feasible time and resource constraints.

correct agreements and recovers from attacks and faults.

Swarm Oracle is a decentralized oracle network composed of autonomous robots that act as hardware oracles, directly sensing and processing data from the physical world. By leveraging swarm robotics' peer-to-peer interactions, decentralization, and self-organization (Dorigo et al. 2014), Swarm Oracle can potentially scale or adapt robots' tasks to flexibly meet the demands of real-world applications (Figure 6.1). Even when composed of simple robots equipped with inexpensive, noisy or biased sensors, Swarm Oracle harnesses the wisdom of the crowd (Galton 1907, Valentini et al. 2015c) through a consensus protocol, aggregating data from multiple robots to reach agreements that are more accurate and reliable than individual robots' sensor readings. Swarm Oracle can, for example, monitor fish stocks, wildfires, traffic conditions, or trash deposits, providing important information required by smart contracts to enact transparent regulations to manage shared resources (Hassan and De Filippi 2021). In this context, Swarm Oracle functions as a Decentralized Physical Infrastructure Network (DePIN) (Lin et al. 2025), retrieving and serving real-world data to blockchains on demand, as part of the growing decentralized economy.

We envision Swarm Oracle as a collective of robots from different stakeholders, a prerequisite for a *trustless* oracle network that prevents biases during the design, manufacturing, or programming of the robots. Achieving consensus in a mobile multi-robot system with unreliable and sparse communication is a major challenge, further complicated by the possibility that stakeholders hold conflicting interests or that malicious actors attempt to manipulate on-chain data for personal gain. Even under these adversarial conditions, Swarm Oracle consensus must be both *fault-tolerant* and *global*—otherwise, discrepancies among the robots could compromise the integrity and reproducibility of the blockchain's deterministic state.

Unfortunately, these requirements are not met by typical swarm robotics consensus methods, where robots gradually adjust their individual beliefs to converge toward an *approximate* collective value, possibly applying local rejection of extreme values to enhance robustness (e.g., the W-MSR rule LeBlanc et al. 2013). While some strategies can withstand adversarial robots (LeBlanc and Koutsoukos 2011, 2012, Zhang and Sundaram 2012b), security hinges on robots' remaining sufficiently connected to honest robots. In sparsely connected robot swarms, these approaches may fail when networks partition—especially if malicious robots target specific neighborhoods. Most critically, they allow for persistent (small) discrepancies in opinions across the swarm, falling short of Swarm Oracle's requirements.

To reach trustless, fault-tolerant and global consensus without requiring external infrastructure, we implement Swarm Oracle through a blockchain hosted by the robots themselves. This second-layer blockchain (also called "sidechain") is adapted to the limitations of the robots, employing a lightweight consensus protocol to avoid

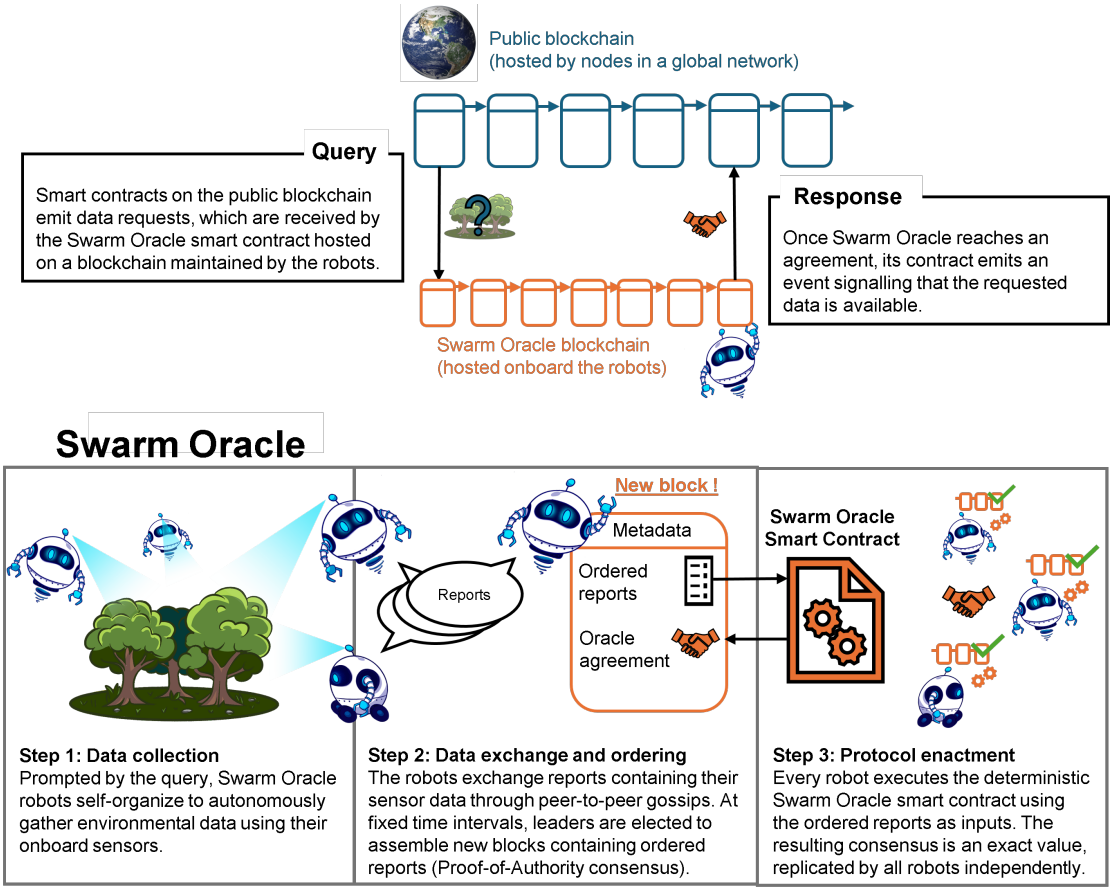


Figure 6.1: Deployment of the Swarm Oracle as a service. In this scenario, the protocol is implemented as a smart contract on a dedicated second-layer blockchain maintained by the robots. By hosting the contract on a blockchain tailored to robots' capabilities, the robots can reach a global (i.e., swarm-wide) agreement from the aggregation of their individual data, with lower latency and transaction costs than a public blockchain deployment.

the high energy demands of protocols based on proof-of-work. The Swarm Oracle protocol, which includes data aggregation methods, is deployed as a smart contract on this blockchain, as illustrated in Figure 6.1. The Swarm Oracle smart contract interacts with external infrastructure through events, receiving queries from public blockchains and publishing consensus results once agreements are reached. A general taxonomy on information exchange methods between oracle networks and smart contracts is available in (Mühlberger et al. 2020). While our framework centers on blockchain as both the application and the deployment vehicle, the trustworthy information produced by Swarm Oracle can also support off-chain decision processes, particularly when transparency and impartiality are required, such as applications for non-profit or government institutions. The Swarm Oracle protocol could equally be deployed as a smart contract on a public blockchain, though this could introduce transaction costs, especially when swarm sizes are large or when robots collect high volumes of data (Dorigo et al. 2024†).

6.2 Methods

The tools employed in this chapter and throughout the thesis are described in Chapter 4, including the robots, the simulation interface, the Ethereum-based blockchain, and the communication and consensus protocols. The following chapter-specific parameter values were used:

- The *block period* B_p is 10 seconds.
- The *maximum communication distance* is 10 cm (in real robots, this is regulated by tuning the infrared signal power, which can be affected by reflections).

The remainder of this section presents the methods specific to this chapter.

6.2.1 Environment and task

The experimental scenario, shown in Figure 6.2, contains three colored features (red, green, and blue), each marked with AprilTags indicating whether they are "valuable" or not. In this study, the red feature is designated as valuable, while green and blue are not. The oracle query under consideration is: "What are the RGB color values of the valuable feature?" This setup illustrates a potential real-world deployment of a Swarm Oracle, composed of a swarm of 12 mobile robots that explore the environment and monitor its physical properties using onboard sensors and peer-to-peer communications. Each robot is equipped with an inexpensive monocular camera used to detect the colors of the features and to read

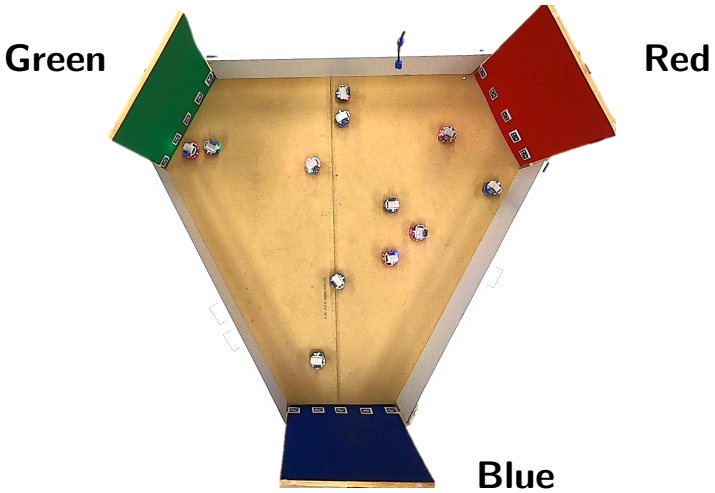


Figure 6.2: Feature perception experimental scenario. The arena contains three colored features that can be detected by the robot cameras from a distance (potentially from any position in the arena if there is a clear line of sight) and used for navigation. Each feature has AprilTags, readable from a short distance (approximately 10 cm), which encode "not valuable" on green and blue features, and "valuable" on red.

the AprilTags. However, color readings are highly noisy due to occlusions, sensor quality, and light conditions. Additionally, individual differences in each robot's color perception introduce systematic biases. As a Swarm Oracle, robots combine their readings to compute an accurate consensus value for the RGB color of the feature (a three-dimensional array of continuous values). Robots can navigate toward visible color blobs, but can only determine whether a feature is valuable or not by reading the AprilTags at close range (approximately 10 cm). Adversary robots can use the green and blue features as distractors to carry out some of their attacks.

6.2.2 The Swarm Oracle – Short description

The design and properties of Swarm Oracle are presented in detail in Section 6.5. Here, we provide a brief overview in the context of our experimental scenario.

Reporting Robots move toward large color blobs detected in the environment. When a robot comes within 10 cm of an AprilTag, it stops and samples the blob's RGB values and reads the AprilTag. We call this measurement an *observation*. The robot then broadcasts a blockchain transaction to nearby robots (within 15 cm) containing: (i) the *observation*, (ii) a token *deposit*, and (iii) a *vote* for the feature's value (AprilTag reading). Through the blockchain, the swarm orders the reports

and maintains a ledger of reputation tokens, with a total circulating supply of T tokens. Sending reports requires that robots deposit a fixed fraction of the reputation tokens they own.

Clustering As reports arrive, a smart contract groups them into clusters using incremental k-means (Pham et al. 2004). Each cluster is associated with a candidate feature value, which we call a *proposal*. The proposal is computed as the reputation-weighted centroid of the observations in its cluster. New clusters are created when the distance between a reported observation and existing proposals exceeds a *clustering threshold* R.

Reaching the quorum A proposal must accumulate a certain quantity of token deposits before a consensus decision is made. The quantity of tokens required depends on two parameters: The *deposit quota* K and the *deposits quorum* q, with both $K, q \in]0, 1]$. The *deposit quota* K sets the number of tokens individual robots must deposit when reporting. To send a report, robot i must deposit Kt_i, where t_i is the number of tokens it owns. Thus, K regulates the maximum number of proposals Swarm Oracle can process in parallel: once $\lfloor K^{-1} \rfloor$ proposals are awaiting a decision, reports that do not match existing proposals are dropped to ensure enough tokens are available to reach the quorum across all active proposals. The *deposits quorum* q sets the fraction of KT that a proposal must reach before a consensus decision is made.

Decision rule and Byzantine fault tolerance Once a proposal reaches qKT in token deposits, Swarm Oracle decides on that proposal's value using a deposit-weighted majority vote among its reports. In our experiments, we set $q = \frac{2}{3}$ to balance *safety* and *liveness* under Byzantine robot behaviors: If q is greater than $\frac{2}{3}$, an adversary with less than one-third of the tokens (i.e., $1 - q$) could cause a deadlock simply by not sending any reports. Conversely, if q is set below $\frac{2}{3}$, the adversary could potentially force an incorrect agreement with less than one-third of the tokens (i.e., $q/2$) by acquiring the majority of deposits in a proposal. As such, selecting $q = \frac{2}{3}$ is a Pareto optimal choice that guarantees both liveness and safety as long as adversaries control no more than one-third of the tokens (Castro and Liskov 2002, Lamport et al. 1982). Figure 6.3 discusses the implications and trade-offs of selecting alternative q values.

Reputation system and resilience To prevent faults from accumulating and degrading performance over long deployments, Swarm Oracle includes a reputation mechanism. After each decision, reports that supported the collective decision are rewarded, and reports that contradicted it are penalized, according to Equation 6.1. Concretely: (i) the deposits from robots in the minority group are redistributed to the majority (improving safety), and (ii) a fixed token bonus, determined by an issuance constant I_c, is minted and distributed among the majority voters

(improving liveness). By using reputation tokens as both participation credentials and weights for future contributions, over time, Swarm Oracle recovers from the damage caused by faulty robots that remain idle or vote against the majority.

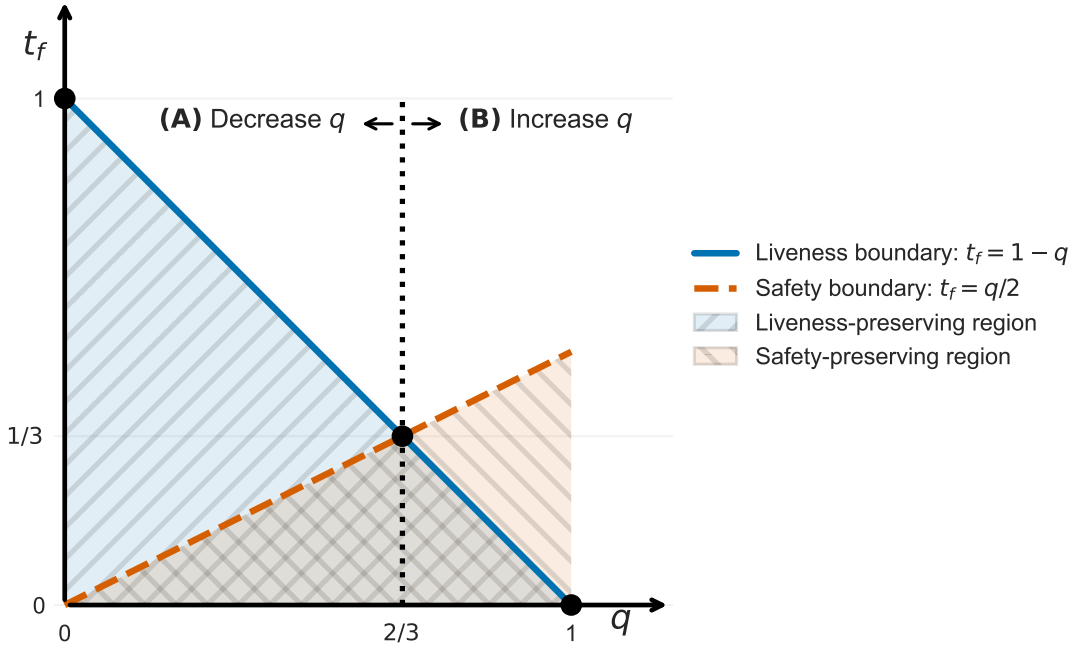


Figure 6.3: Safety and liveness design space. The y-axis shows the fraction of tokens held by Byzantine (faulty) robots, and the x-axis shows the deposits quorum required to decide on a proposal in Swarm Oracle. We analyze how varying q around the value $q = \frac{2}{3}$ affects safety and liveness. **(A)** Decreasing q shifts the liveness boundary upward (blue line), enlarging the liveness-preserving region, since Byzantine robots require a larger token fraction t_f to stall agreements. However, it simultaneously shifts the safety boundary downward (dashed orange line), as fewer tokens are sufficient to force an incorrect decision. **(B)** Increasing q has the opposite effect: the safety-preserving region expands, but the liveness-preserving region shrinks, since a smaller Byzantine token share can stall consensus. In adversarial settings, $q = \frac{2}{3}$ is optimal because it balances safety and liveness, ensuring that an attacker must control the largest possible fraction of tokens to either force an incorrect decision or block progress. In cooperative swarms, where Byzantine robots do not coordinate to manipulate consensus, decreasing q may be desirable to enable faster and less costly agreement while maintaining an acceptable level of safety.

6.2.3 Robot controllers

Our robots follow a routine that mainly consists of *exploring* the environment to find features and initiate new proposals, or *validating* existing proposals in the smart contract. To achieve this, the robots detect the colors of features in the environment and navigate towards them to measure their RGB values and read the AprilTags. The robots' state machine is illustrated in a flowchart in Figure 6.4.

1. `Query` The robot queries the current proposals from the smart contract. If there are proposals it has not yet validated, it randomly selects one and transitions to the `Validate` state; otherwise, it transitions to the `Explore` state.

2. `Validate` The robot searches the environment for features and navigates to the one that is the closest match to the proposal to be validated. Upon arrival, it transitions to the `Report` state, or returns to the `Query` state after 100 s.

3. `Explore` The robot searches the environment for any feature and then moves towards it. Upon arrival, it transitions to the `Report` state, or returns to the `Query` state after 100 s.

4. `Report` Once the distance to the AprilTag attached to a feature is smaller than a threshold, the robot reads the AprilTag and the average RGB value of the largest color contour in its camera vision. Then it sends a report through a blockchain transaction and returns to the `Query` state.

6.2.4 Byzantine faults and attacks

We use the term *Byzantine* to refer to both faults and attacks, emphasizing that Swarm Oracle makes no distinction between the two in its design. To evaluate Swarm Oracle's fault tolerance, we systematically introduce *Byzantine attackers* into the swarm. The attackers are robots with deliberately modified behaviors designed to degrade Swarm Oracle's safety and/or liveness. In each experiment, all attackers employ the same modified behavior, since a coordinated strategy (collusion) maximizes their chances of success. We investigate four distinct attack types, each targeting a critical aspect of the system's operation:

Safety attack The attackers report green and blue features as "valuable" and red as "not valuable".

Liveness attack The attackers abstain from voting.

Combined attack The attackers combine the two previous strategies: they report for green and blue features, but abstain from reporting red.

Physical attack The attackers attempt to disrupt other robots' camera readings by physically blocking their line of sight and activating blue LEDs (Figure 6.5).

Robots also exhibit *Byzantine faults* that occur unintentionally, i.e., without explicit fault injection. Faults can affect safety, for example when obstructions or unusual lighting conditions lead to incorrect proposals. They also affect liveness, for example when robots become stuck due to low-quality obstacle sensors or wheel actuators. Unlike attacks, Byzantine faults are considered transient, since we expect that the robots eventually recover and resume normal operations. If persistent faults occur (e.g., broken motors, damaged sensors or disconnected cables), we halt the experiment and repair the affected robots to prevent faults from accumulating and biasing results over time.

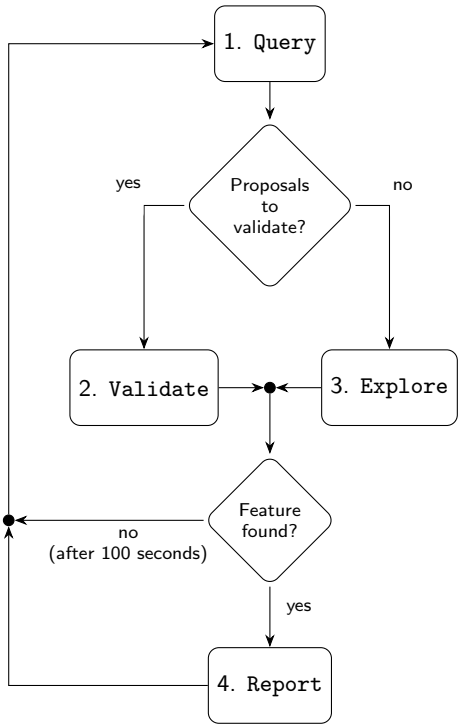


Figure 6.4: Robots' state machine flowchart.

Figure 6.5: Bodywall formed by the physical attackers. Results show a marked increase in costs with 3–4 physical attackers. At these numbers, the attackers can form effective barriers, temporarily blocking access to the feature.

6.2.5 Experimental setup

The experiments were conducted in two batches: *short-run* and *long-run*. The short-run experiments are performed on a physical swarm of 12 Pi-puck robots (Millard et al. 2017) and terminate once the first agreement on the valuable feature is reached. We study the effect that the clustering threshold R and the number of Byzantine attackers f have on the quality and costs of agreements (Table 6.1). The long-run experiments are simulated in the ARGoS simulator to let us study how robots' reputation evolves over multiple agreements, while varying two parameters: the deposit quota K and the issuance constant I_c (Table 6.2)

In both experiments, the deposits quorum q is kept at the fixed value of $\frac{2}{3}$, as we want to maximize Swarm Oracle's robustness to Byzantine attackers.

		Clustering Threshold R (with f = 0)				**# of Byzantine Attackers f (with R = 60)**				
		20	40	60	80	0	1	2	3	4
Type of Attack	Safety	-	-	-	-	-	10	10	10	10
	Liveness	-	-	-	-	-	10	10	10	10
	Combined	-	-	-	-	-	10	10	10	10
	Physical	-	-	-	-	-	10	10	10	10
	No attack	10	10	10*	10	10*	-	-	-	-

Table 6.1: Short-run experiments (real robots). The table shows the number of trials per configuration. The star symbol (*) indicates when parameter configurations are repeated. In total, we performed 200 trials over a total duration of 22 h 32 m.

		Deposit Quota K (with $\mathbf{I_c = KT_0}$)		**Issuance Constant $\mathbf{I_c}$ (with $\mathbf{K = 1/3}$)**	
		1	1/3	0	KT_0
Type of Attack	Safety	40	30*	30	30*
	Liveness	40	30*	30	30*
	Combined	40	30*	30	30*
	Physical	40	30*	30	30*

Table 6.2: Long-run experiments (in simulation). The table shows the number of trials per configuration. The star symbol (*) indicates when parameter configurations are repeated. In all long-run configurations we use $f = 4$ Byzantine attackers and $R = 60$ clustering threshold. T_0 is the initial token supply. In total, we performed 400 trials over a total duration of 278 h 52 m.

(A) Independent variables

Clustering threshold The clustering threshold R is an internal parameter of the k-means clustering algorithm that defines the minimum required distance between a new observation and existing cluster centroids (i.e., proposals) for a new cluster to be initiated. In our experiments and real deployments, noisy camera readings can lead to faults, such as incorrect votes or robots failing to find proposals. To examine the impact of faults caused by noisy measurements, we could keep a fixed R and add noise or biases to the readings gathered by the robots. However, this approach would require artificial noise models, which are less interesting to analyze compared to the natural noise and color miscalibration already affecting the robots' readings. Instead, we run a set of short-run experiments where we vary the value of R, to analyze the impact of faults caused by erroneous classifications due to noise. This is showcased in Figure 6.6.

Number of Byzantine attackers In the short-run experiments, we also vary the number of Byzantine attackers f. One of the main features of employing a global protocol is the certainty of correct agreements provided Byzantine robots remain below the $\frac{1}{3}$ threshold. In Figure 6.7 we show that our method resists all attacks when up to 4 Byzantine robots are present (in a total of 12 robots).

Deposit quota The deposit quota K establishes the percentage of reputation tokens that each robot must deposit when submitting a report. Indirectly, it regulates the maximum number of pending proposals: if $K = 1$, only one proposal can remain pending, otherwise $\lfloor K^{-1} \rfloor$ proposals can remain open. Notably, this has an effect on the dynamics of the token redistribution between the robots. In Figure 6.9 we can compare $K = \frac{1}{3}$ at the top, with $K = 1$ at the bottom. With

$K = \frac{1}{3}$, the redistribution of tokens is less steep (i.e., more time and agreements are required for Byzantine robots to lose tokens).

Issuance constant The issuance constant I_c is employed in the first term of Equation 6.1 to generate new tokens that are rewarded to the robots that participated in the voting round. This generates an inflationary effect that penalizes robots that abstain from voting, and improves the liveness of the protocol. If the issuance constant is set to zero, the robots performing the attack on liveness are not penalized (Figure 6.9). During a real deployment, this could lead to the accumulation of faults due to more robots becoming stuck or disabled, potentially compromising the protocol's liveness.

(B) Evaluation metrics and baseline

Consensus error The consensus error is the Euclidean distance between the agreements generated by Swarm Oracle and the average value of all red observations made by non-attacking robots during the short-run experiments. Since the number of experiments was large, we consider the average values to be a good estimate of the RGB values of the features' colors. A point cloud plot of these observations is shown in Figure 6.8. Using an estimated value is needed, since the true RGB values of the color panels are unknown (they depend on the camera sensors, their parameters and lighting conditions).

Time to consensus This is the time it took, in minutes, until an agreement on a "valuable" feature is reached and recorded on a blockchain block. In the short-run experiments, it consists of the time to reach the first agreement, while in the long-run experiments we reset the clock each time an agreement occurs.

Reports to consensus Similarly to time, this is the number of reports sent by the non-attacking robots before an agreement occurs. While time is an intuitive metric, the number of reports is invariant to factors such as the speed of the robots and how fast they can make observations, which are respectively related to, for example, the battery levels (which change as the experiments progress), and to the environment and observation spaces (which change depending on the scenario).

Baseline Distributed agreements in robotics are typically achieved through approximate local consensus, where each robot maintains an individual opinion that it shares with its neighbors. In such algorithms, the exact moment at which consensus is reached is not well defined, as agreement is determined by a threshold rather than by a discrete decision event.

6.3 Results

In this section we report the results for the short-run experiments in real robots, and for the long-run experiments in simulation.

6.3.1 Short-run

In the short-run experiments, we vary the clustering threshold R and the number of Byzantine attackers f to analyze the system's performance and robustness to different types of attack. We analyze the costs, defined as the time and number of reports to reach consensus, and the consensus error, measured as a Euclidean distance in RGB space.

(A) Clustering threshold analysis

Figure 6.6 shows that setting the clustering threshold R too low leads to an increase in both consensus error and costs. This happens because the reports from robots observing the same feature fall into different clusters, increasing the consensus error and costs, both in time and reports. On the other hand, if R is too high, there is a risk that observations of distinct features are clustered together. This may pose a safety threat, for example, if noisy observations of red are matched to green or blue proposals, resulting in an incorrect consensus. It is therefore safer to adopt a conservative threshold, though not excessively so, to maintain reasonable costs. Based on these results, we fix the clustering threshold at $R = 60$ for all subsequent experiments.

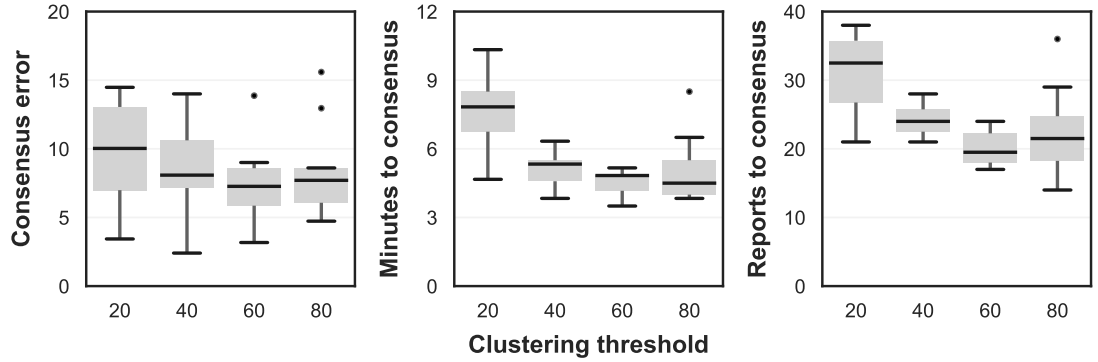


Figure 6.6: Consensus error and costs versus clustering threshold. The plots suggest that, in this instance, there is an ideal value for which the error and costs are lowest. We use this value ($R = 60$) for the other experimental configurations. The consensus error is a Euclidean distance in the RGB space.

(B) Robustness analysis: Performance and costs

Having established a reasonable clustering threshold, we study how Swarm Oracle performs when we introduce attacking robots. In Figure 6.7, the baseline (the arithmetic average of all reports) performs poorly, with high error under safety and combined attacks due to the integration of incorrect observations. It also shows high variance, since in some runs attackers detect colors less often and thus send fewer reports. However, since liveness and physical attackers do not send incorrect reports, consensus error is unaffected for these attack types.

On the other hand, Swarm Oracle agreements remain as accurate as the attack-free results (the consensus error stays below 20 RGB distance units). This demonstrates that with fewer than 4 attackers (one third in a swarm of 12), Swarm Oracle is capable of rejecting all incorrect reports. Although there is a chance that unintentional faults compound with attacks and cause incorrect agreements, in particular when there are 4 attackers, this did not occur in the short-run experiments (it did however occur in the long-run experiments).

Figure 6.7 also shows that consensus costs—both in terms of minutes and reports—increase with the number of attackers. However, the reasons behind these increases are fundamentally different for each type of attack. Facing *safety attacks*, the non-attacking robots must divide their efforts between validating correct proposals and rejecting incorrect ones. As a result, both the time and reports to reach consensus increase. In contrast, during *liveness attacks*, attackers do not submit incorrect proposals, so the number of reports remains constant. Consensus time, however, grows at a rate similar to safety attacks because, without attackers' participation contributing towards reaching the $q = \frac{2}{3}$ quorum, the system is more vulnerable to unintentional faults that temporarily prevent robots from reporting (e.g., getting stuck or not locating the feature due to obstructions or traffic congestion). The *combined attack* compounds the effect of both previous attacks, resulting in the highest overall costs in terms of time and reports. For this reason, our reputation system penalizes each attack type independently, leading combined attackers to lose reputation faster (see long-run results). Finally, in *physical attacks*, attackers activate their blue LEDs and block the red feature, targeting both safety and liveness by disrupting other robots' movement, line of sight, and perception. This results in consensus time comparable to combined attacks, especially when 3 or 4 attackers form effective body-walls (Figure 6.5). Once at the feature, however, physical attackers stay static and issue fewer reports, leading to lower reporting costs.

Figure 6.8 shows the point cloud of all observations made by robots during the short-run experiments and the agreements reached (accepted proposals) with a clustering threshold $R = 60$.

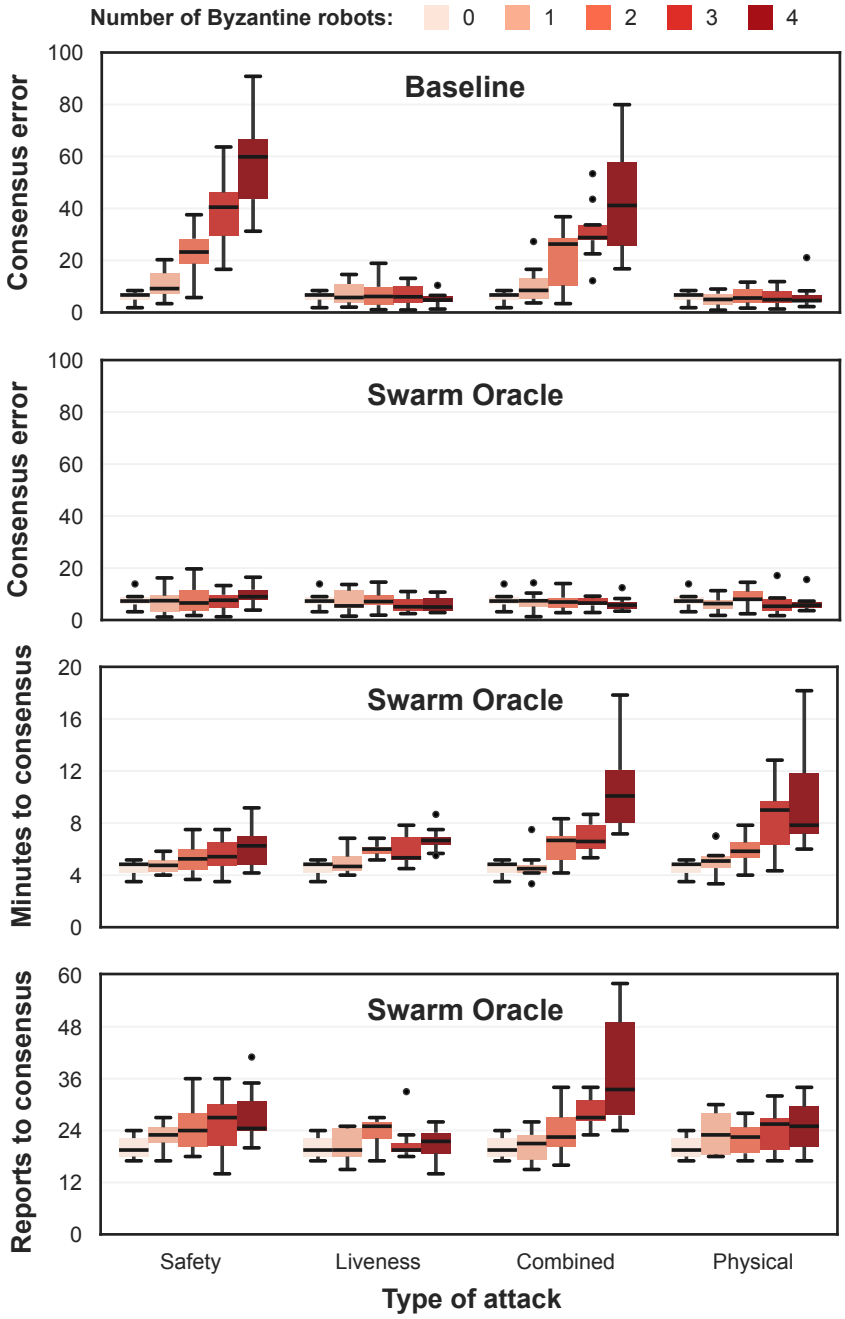


Figure 6.7: Short-run consensus error and costs. The top plot shows the baseline results, while the remaining three plots show the results obtained with Swarm Oracle.

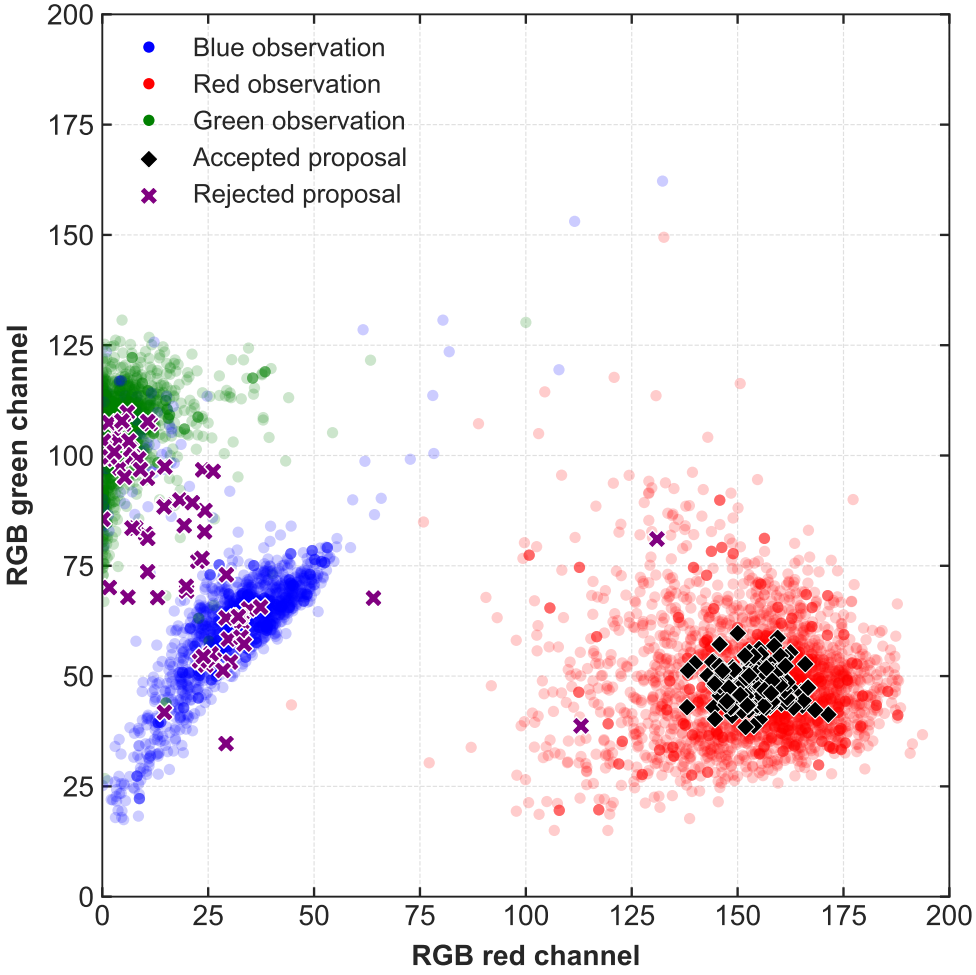


Figure 6.8: Short-run observations and proposals point cloud. The plot shows two dimensions for clarity. The proposals initiated due to attacker observations are rejected with 100% success rate (purple crosses). Some noisy red observations due to unintentional faults led to two rejections. The point cloud shown corresponds to the clustering threshold $R = 60$. Increasing R would make the system more inclusive of noisy observations, but also increase the variance in accepted proposals.

6.3.2 Long-run

In the long-run experiments, simulated in ARGoS (Pinciroli et al. 2012), we study how robots' reputation evolves over multiple agreements. The simulations accurately replicate key physical aspects, including collisions, occlusions, motion, and communication limits. To simulate noisy color perception, each time a robot detects a feature, it randomly samples an RGB observation of that feature made by a corresponding real robot (Figure 6.8).

(A) Reputation dynamics

Long-run experiments are initiated with 4 attackers holding exactly $\frac{1}{3}$ of the total reputation tokens. We chose to study the system at this limit condition, where Swarm Oracle may fail if any additional robot incurs faults, even if unintentionally. Such failures occurred in 5 of the 400 long-run trials, in which attackers ultimately acquired more than 60% of all tokens. These trials are excluded from the plots, as they represent an operating regime beyond Swarm Oracle's intended design. Importantly, once an initial correct agreement is reached, reputation tokens are redistributed, making it increasingly difficult for the attackers to succeed.

In Figure 6.9, although all robots incurring faults may lose reputation, the attackers are the most penalized, ending with a significantly lower proportion of the total tokens. The rate at which they lose tokens changes with the parameters regulating the reputation system, namely, the deposit quota K and the issuance constant I_c. With $K = 1$, robots can only work on one consensus proposal at a time, leading to faster agreements and redistribution of tokens. Setting $K = \frac{1}{3}$ initially leads to slower agreements, however, it allows robots to physically distribute themselves in the environment when contributing to different proposals, resulting in more efficient and robust operation, and in more consistent intervals between subsequent agreements. While safety attackers lose their deposits after a successful vote—according to the second term in Equation 6.1—liveness attackers are only penalized when $I_c > 0$, through the first term in Equation 6.1 which dilutes the stake of non-participating robots by issuing new tokens. Finally, different attacks lead to different rates of token loss. Since safety attackers do not abstain from voting, agreements occur more frequently, leading to a faster redistribution of tokens. The combined and physical attackers are penalized by both terms in Equation 6.1, however, since they succeed at slowing down agreements (Figure 6.7), their loss of reputation is also slower. Across all parameter settings, non-faulty robots end the experiments holding the majority of tokens, thereby restoring Swarm Oracle's safety and liveness margins and, as shown next, reducing consensus costs.

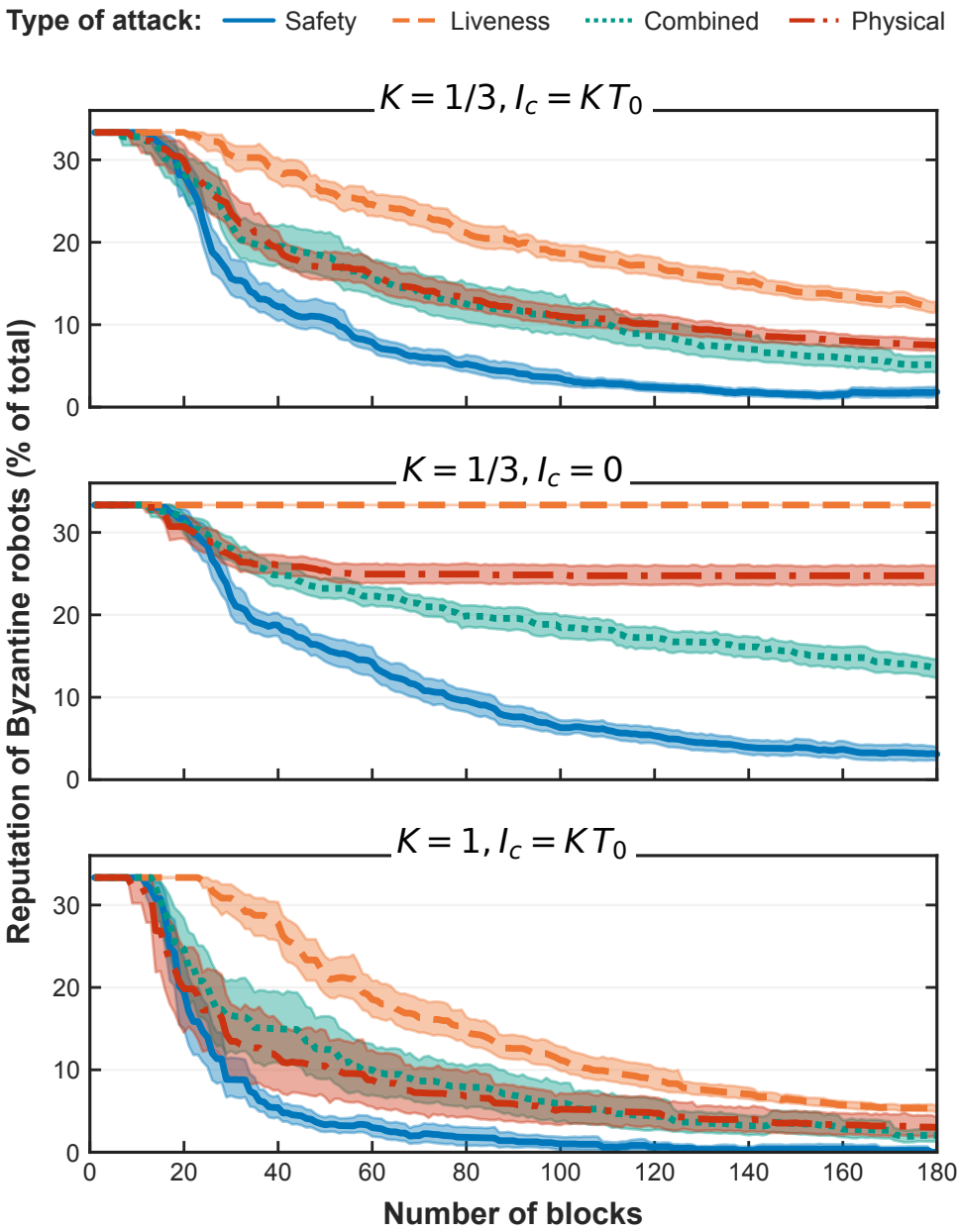


Figure 6.9: Fraction of reputation tokens held by attacking robots. Reputation updates occur as new blocks are added to the blockchain, approximately every 10 seconds.

(B) Resilience: Autonomous recovery from faults

Figure 6.10 shows that the cost of reaching agreements decreases over time, as the reputation system gradually mitigates the negative impact of the attackers. After approximately five agreements, attackers achieve low reputations, resulting in a reduction of the average time to achieve consensus, from 6 minutes (first agreement) to 3.7 minutes. The same trend is observed in the number of reports, which decreases from an average of 22 to 13. These results highlight Swarm Oracle's self-healing ability, autonomously returning to an efficient operating state.

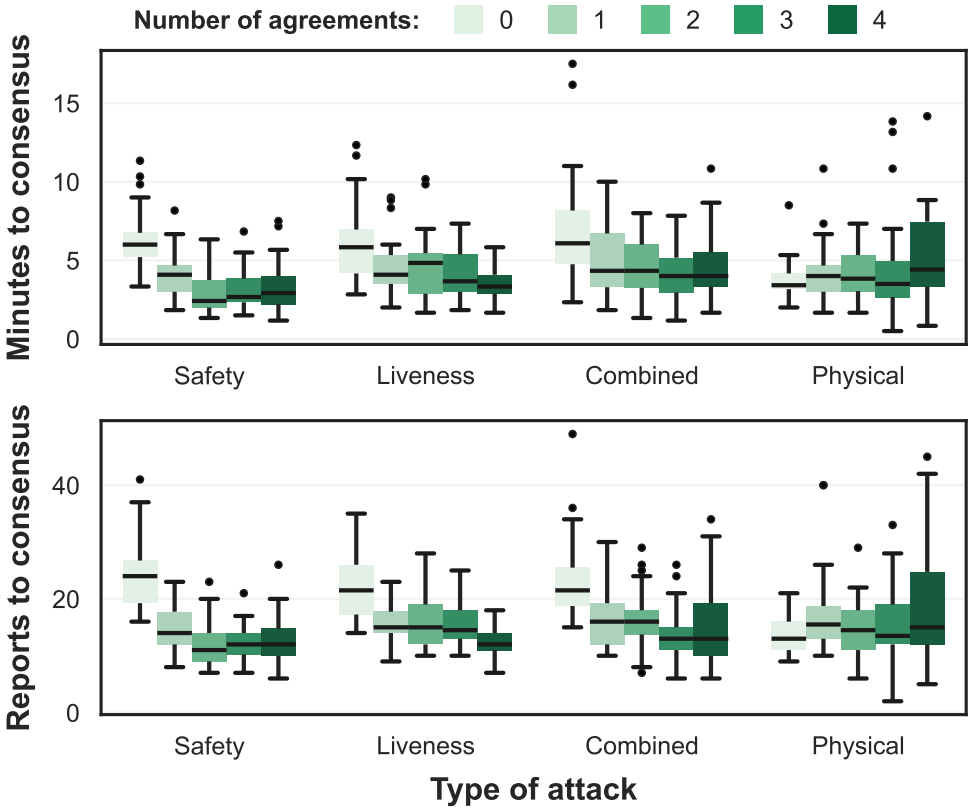


Figure 6.10: Long-run autonomous recovery from faults. As reputation tokens are redistributed among non-faulty robots, the costs to achieve consensus tend to return toward their values when no Byzantine robots are present.

6.4 Discussion

Our experiments showcase the fault-tolerance of the Swarm Oracle and its limits when under extreme attacks. In the short-run experiments, one or two attackers only modestly increased consensus time (less than 50%). However, with three or four attackers, consensus time increases sharply, and with four attackers, even a single robot with unintentional faults could potentially delay consensus or lead to an incorrect agreement. Although no incorrect agreements occurred in short-run experiments, they appeared in five long-run trials. Importantly, once an initial correct agreement is reached, our reputation system reduces the impact of attackers, showcasing the system's self-healing ability: over time, the Swarm Oracle recovers and progressively neutralizes attacks.

The following subsections discuss the scalability of Swarm Oracle, its potential vulnerabilities and their mitigation, and considerations for its deployment.

6.4.1 Scalability

Achieving scalable fault-tolerance is one of Swarm Oracle's key advantages. Since the protocol's security relies on a global condition (in contrast with swarm robotics' local ones), larger swarms can tolerate a greater absolute number of faulty robots: While a swarm of 12 robots tolerates 4 faulty robots, a swarm of 48 tolerates 16. If we assume that 1 in 10 robots are faulty, the smaller swarm can withstand an additional 3 faulty robots, in comparison with 11 for the larger swarm. Larger swarm sizes also make attacks more challenging: attackers must coordinate a larger number of robots, which becomes increasingly difficult given the network size and spatial distribution of the robots.

Hardware costs also scale well, since the number of reports increases linearly with the number of robots, and communications between robots are local and very sparse. These costs are further reduced by our use of a cost-efficient blockchain protocol and dual-layer communications (infrared broadcasts for peer identification, and Wi-Fi for exchanging reports and block metadata). This was discussed in depth in our proof of concept study, presented in Chapter 5, where we showed that the hardware costs and blockchain consensus remain scalable for swarm sizes between 8 and 24 robots, and with up to 120 simulated robots.

The main scalability challenge arises from the requirement that a supermajority of the robots participate in consensus (more precisely, the holders of $q = \frac{2}{3}$ of the reputation tokens). This becomes costly when robots and observation areas are either widely dispersed, leading to large physical costs to generate reports, or overly concentrated—potentially leading to physical interference between robots. To mitigate this, Swarm Oracle could be deployed in smaller committees within a

larger swarm. The committee size can be tuned to match application requirements, enabling fault-tolerance to scale in accordance with the levels demanded by the application. Our implementation also includes the possibility of parallelization of consensus (through the K parameter), improving scalability by enabling robots to work on multiple consensus proposals at a time. Another option is to reduce the deposits quorum q, enabling faster agreements requiring participation from fewer robots—however, reducing the system's robustness to Byzantine robots.

6.4.2 Possible vulnerabilities and their mitigation

Denial-of-Service The deposit quota parameter K limits the number of pending proposals, making it possible for attackers that continuously report incorrect observations to occupy all available slots. In theory, liveness is not compromised since asynchronous communication prevents attackers from reporting at the exact time a slot becomes available. However, an attacker could willingly sacrifice all of their reputation to continuously attempt the attack, since reporting only requires a deposit proportional to the robot's reputation. One solution is to not allow low-reputation robots to vote, effectively excluding them from Swarm Oracle until they are repaired or secured. Alternatively, a minimum reputation could be required to open new proposals, allowing robots with low reputation to still participate in open proposals and regain reputation through useful contributions.

Copycats To ensure the decentralized and trustless execution of the Swarm Oracle smart contract, it is required that observations, as well as robot reputations, are public so that they each compute the same weighted average. However, this transparency introduces a vulnerability: a malicious robot could wait to receive reports from others, and copy or average the observations from the most reputable peers to unfairly gain reputation. This type of attack is known in decentralized finance (Barczentewicz et al. 2023), where trading bots copy the transactions of the most successful traders. Since hiding the reputations or observations is not possible, a potential solution is to instead conceal the votes using cryptography (Van Calck et al. 2023[†]). In this approach, robots submit encrypted votes, which are only revealed once the proposal reaches its deposit threshold (after which no more reports are accepted). Robots that fail to reveal their vote would be penalized (e.g., by losing their deposit), and if a missing reveal is decisive for the outcome, the round is nullified.

Unobservability Byzantine robots may generate proposals based on fictitious observations, which are not present in the environment. In such cases, simply relying on robots to report their own observations is insufficient, as unobservable proposals may remain pending. In the worst case, if all available proposal slots are unobservable, Swarm Oracle could reach a deadlock. The simplest mitigation

strategy is to automatically reject proposals that remain pending after a fixed number of minutes, blocks, or reports. Another is to have the robots vote negatively if they fail to observe anything matching the proposal within a reasonable time. In our experiments, we adopted a variant of this approach: robots attempt to *validate* existing proposals by searching the environment for a fixed amount of time, and then traveling towards the observation that is the closest match. The resulting report includes the index of the proposal they aim to validate. If the report is not matched to the intended proposal by the clustering algorithm, then the reputation deposit is counted until the qKT threshold is reached, after which the proposal is dropped.

Physical Attacks A key aspect of mobile robotics systems is that robot interactions occur not only through communication, but also physically. While our protocol protects against attacks exploiting communication, physical attacks also warrant attention. In our robotics context, physical force may be used to interfere with other robots or manipulate the environment, inducing faults in otherwise non-faulty robots to degrade system performance and avoid accountability. Our experiments showed how physical attackers caused degradation comparable to that of combined attackers, but lost reputation at a slower rate. Since physical interactions occur outside of digital communications, ensuring fault-tolerance remains an open problem. Interestingly, one solution could be to use Swarm Oracle itself to let the robots report observed physical attacks and agree on the identity of the attackers. These robots could then be permanently expelled from Swarm Oracle, and the information could be passed to law enforcement agencies or to the public blockchain, to hold robot stakeholders accountable.

6.4.3 Swarm Oracle deployment

We deployed the Swarm Oracle on a group of 12 robots, each self-hosting an Ethereum (Buterin 2014) instance. Although a blockchain is not strictly necessary, Swarm Oracle requires methods to ensure an ordering of the reports, track reputation tokens, and prevent double-spending attacks (in which one robot could use its reputation tokens multiple times). Additionally, the robots need to synchronously and correctly execute the deterministic logic that underlies the Swarm Oracle protocol and its reputation system. These requirements are naturally fulfilled by blockchain smart contracts, which are widely available through open-source blockchains.

Blockchains also help to reduce the communication overhead and storage requirements by aggregating information into blocks generated at specified time intervals (every 10 seconds on average in our experiments). Communication is very efficient since robots exchange reports through sparse and local gossip, and blocks require

minimal bandwidth as they only contain essential metadata and the ordering of reports. Finally, the linear structure of the blockchain results in predictable storage growth, and the immutability of the cryptographic links between blocks allows robots to safely prune historical data to further save space.

Swarm Oracle could also be deployed using other distributed ledgers, such as directed acyclic graphs (Tran et al. 2019) or blocklists (Wardega et al. 2023), or through protocols tailored for swarm robotics, such as SwarmMesh (Majcherczyk and Pinciroli 2020) or Buzz (Pinciroli and Beltrame 2016). However, these alternatives do not provide the security and convergence guarantees of blockchain-based protocols, and other potential advantages are yet to be demonstrated in practice.

Another alternative is to deploy the Swarm Oracle smart contract on a public blockchain (Dorigo et al. 2024[†]). This reduces the communication and computational burden on the robots, while leveraging the decentralization and automation of smart contracts hosted on public blockchains. We tend to favor the local approach, where the Swarm Oracle protocols are maintained by the robots through a private blockchain, and only the consensus results are periodically uploaded to a public blockchain. This strategy preserves autonomy and fault-tolerance, while minimizing reliance on external connectivity and reducing transaction costs on public blockchains.

Swarm Oracle empowers robots from different stakeholders to pool their capabilities to provide reliable and transparent data as a service. Unlike traditional swarms, Swarm Oracle protocols are designed to withstand adversarial behaviors, enabling collaboration without the need for trust. The data gathered by Swarm Oracle can be provided to external applications, but it can also be employed internally by the swarm. Recent research increasingly explores how robot swarms can employ blockchains to support self-organization, governance and economic participation, while identifying the oracle problem as a fundamental challenge (Castelló Ferrer 2018, Dorigo et al. 2024[†], Peña Queralta et al. 2023). Swarm Oracle can also be used to securely update distributed data structures maintained by the robots, other than blockchain, such as tuple-spaces (De Nicola et al. 2020, Majcherczyk and Pinciroli 2020, Pinciroli et al. 2016), environment maps (Birk and Carpin 2006, Kegeleirs et al. 2021, Lajoie and Beltrame 2024), coordinate systems (Jones and Hauert 2025, Pluhacek et al. 2025), directed acyclic graphs (Keramat et al. 2023, Tran et al. 2019), blocklists (Wardega et al. 2023), and Turing-complete state machines (Pinciroli and Beltrame 2016).

6.5 The Swarm Oracle – General description

Swarm Oracle consists of a group of robots, denoted as $\mathcal{S}$. The robots, operating in a real-world environment, individually collect and process data using their

own sensors and computational resources. The data obtained by the robots (e.g., readings from their sensors) forms an *observation space,* denoted as Ω. The goal is to develop a consensus protocol that lets robots agree on values in Ω while meeting the safety and liveness requirements: the protocol always produces correct and consistent agreements (safety), and every non-faulty robot eventually decides on a value (liveness). The *consensus set* $\mathcal{L} \in pow(\Omega)$ is the set of all agreements reached by the robots.

As an example, if the robots' observations consist of GPS coordinates marking the positions of valuable resources, then $\Omega = \mathbb{R}^2$, and $\text{pow}(\Omega)$ represents any possible set of positions on the Earth's surface. In this example, the goal of the consensus protocol is to enable the robots to continuously update the set $\mathcal{L}$ with accurate resource positions.

6.5.1 Byzantine fault model

Robots operating in the real-world are susceptible to a wide range of potential faults. While some faults can be anticipated and mitigated through careful design of the robots, their behaviors, and their communication protocols, others cannot: a robot may develop faults causing unpredictable and erratic behavior, or multiple robots may collude to attack the system if controlled by a malicious adversary. Such faults are commonly referred to as Byzantine faults (Lamport et al. 1982).

We consider a Byzantine fault model because distinguishing unintentional from malicious faults is often impossible: a robot that fails to send observations due to broken wheels is indistinguishable from one that deliberately stops, and a robot that sends incorrect observations because of a dirty sensor is indistinguishable from one that does so to mislead the consensus. Malicious robots may, however, coordinate their actions to carry out stronger attacks, timing them to exploit faults present in other robots. Therefore, the developed protocol must be *Byzantine fault-tolerant* to guarantee correct and continuous operation in real-world deployments, even in the presence of malicious attacks and unintentional faults.

Faults can also differ significantly in terms of duration: robots may experience temporary faults and subsequently recover, but they can also become permanently disabled or remain under an adversary's control for extended periods of time. Since the multi-robot system is assumed to operate autonomously, the protocol should include mechanisms to mitigate the long-term negative effects of permanent or persistent faults, which could otherwise degrade performance and ultimately lead to system failure. To address this issue, we integrate a *reputation system* into the protocol that weighs each robot's contributions based on its performance, eliminating the need for a repair technician or external authority during operation. Importantly, our reputation system does not exclude faulty robots: they are allowed to continue participating and may regain reputation if they recover from their

faults.

6.5.2 Protocol

The number of robots in the Swarm Oracle is $N = |\mathcal{S}|$, and each robot has an associated digital identity $i = 1, \ldots, N$, which controls t_i reputation tokens. The reputation tokens of each robot, and consequently the total amount of reputation tokens $T = \sum_{i=1}^{N} t_i$, are globally known.

Let's initially assume that each robot can obtain perfect observations, for instance, in the case where their sensors are noise-free. Even though this condition is not realistic and will be later relaxed, it helps to establish and understand the functioning of our oracle consensus protocol. Robots locally broadcast structured reports, which are disseminated across the swarm. A report r contains an observation $r.o \in \Omega$, the robot identity $r.i$, a deposit of reputation tokens $r.d$, and a vote, signaling acceptance or rejection, $r.v$. Through the blockchain, Swarm Oracle ensures a global ordering of the reports and that they are structurally valid, creating an array $\boldsymbol{r}$ of ordered reports. Then, through the Swarm Oracle smart contract, the following deterministic logic is applied to the array $\boldsymbol{r}$:

1. The reports are grouped according to their observations. Each group of observations generates a *proposal*.
2. Once a proposal reaches a cumulative value of deposits greater than $q\,T$, the voting decision takes place, where q is the deposits quorum.
3. An absolute majority is required to determine whether the proposal is accepted or rejected using the identities, deposits, and votes in the reports associated with the proposal.
4. A reputation system redistributes the deposited tokens based on the outcome of the voting round.
5. The reports that have been used in the voting round are removed from $\boldsymbol{r}$.
6. If the proposal is accepted, it is added to the consensus set $\mathcal{L}$.

Byzantine fault-tolerance By setting $q = \frac{2}{3}$ we guarantee the maximum level of Byzantine fault-tolerance: safety and liveness are guaranteed, as long as faulty robots control less than one-third of the total token supply. This choice is Pareto optimal: any deviation from $q = \frac{2}{3}$ would decrease the Byzantine fault-tolerance of the system: By increasing it, the Byzantine robots could cause a deadlock with less than $\frac{1}{3}$ of the reputation tokens by simply not sending reports; and by decreasing it, the faulty robots could force incorrect agreements with less than $\frac{1}{3}$ (i.e., half of

$\frac{2}{3}$) of the reputation tokens by obtaining the absolute majority stake in a voting round. In non-adversarial environments, it may be warranted to choose $q < \frac{2}{3}$. However, since Swarm Oracle envisions outdoor deployments with robots from different stakeholders, we restrict the discussion to the $q = \frac{2}{3}$ case.

Parallelization In some applications, most notably those in swarm robotics, it may be inefficient to require a $\frac{2}{3}$ supermajority to reach agreements, since some robots may be slower than others when submitting their votes. To improve efficiency by enabling the robots to work in parallel, we introduce the *deposit quota* parameter, denoted $K \in]0, 1]$, which establishes the number of reputation tokens that a robot i must deposit with its reports ($r.d = K\, t_i$), as well as the cumulative deposits required before a voting round occurs ($\frac{2}{3}\, K\, T$). Note that this parameter indirectly regulates the maximum number of pending proposals: there can only be $\lfloor K^{-1} \rfloor$ pending proposals at a given moment, otherwise, there would not be enough circulating reputation tokens to reach the cumulative deposits requirement on all of them, and the system could come to a deadlock.

With this in mind, steps 1 and 2 in the logic above are adjusted as follows:

1. The reports are grouped according to their observations, up to a maximum of $\lfloor K^{-1} \rfloor$ groups. Each group of observations generates a *proposal*.

2. Once a proposal reaches a cumulative value of deposits greater than $\frac{2}{3}\, K\, T$, it enters a voting stage.

When observation tasks can be efficiently performed in parallel—for example, when observations result from unpredictable events, such as finding an object during random walks in large environments—then a low value of K may be desirable so that the robots contribute towards multiple proposals simultaneously, exploiting parallelization and reducing physical interference. Conversely, if observations require dedicated effort (e.g., searching a specific area), or when the interference between robots is low, then higher values for K may be warranted, allowing robots to concentrate on validating existing proposals and potentially accelerating the consensus process. The deposit quota K is, therefore, a parameter that can be tuned as a function of the oracle's application context.

Dealing with noise To have the robots reach agreements despite noisy observations, the individual observations of each robot stored in the shared array $\boldsymbol{r}$ are aggregated, e.g., by applying filtering and/or averaging functions. Our proposed method employs a clustering algorithm to group robot reports based on a similarity score (e.g., a distance function). In doing so, the reports from robots will be matched to the closest proposal. If a report is not matched to existing proposals, a new cluster is created as long as the number of clusters does not surpass $\lfloor K^{-1} \rfloor$ (otherwise, the observation is dropped and the reputation tokens are refunded).

An aggregation function is then employed to transform the reports in a cluster $\mathcal{C}_j$, $j \in \{1, \ldots, \lfloor K^{-1} \rfloor\}$ into a proposal $p_j \in \Omega$.

The clustering algorithm, the aggregation function and their parameters are chosen as a function of the oracle's application context and, in particular, of the observation space. In our experiments, we employed incremental k-means clustering (Pham et al. 2004), with a simple rule for incrementing the number of clusters: a new cluster is generated when the distance between a new observation and all existing proposals exceeds a threshold R. The aggregation function generates proposals that correspond to the average value of the observations, weighted by the deposits associated with each report in a cluster. In Section 6.3 we explore the effects of varying the clustering threshold R.

Autonomous recovery from faults In computer science, faults are often treated as temporary events, making dedicated detection and mitigation unnecessary, with prevention and tolerance preferred instead (Anderson and Knight 1983). In robotic systems, however, various faults may occur that robots cannot automatically recover from. While in controlled environments, such as warehouses, fault detection may suffice—since faulty robots can be serviced by technicians or removed from operation—in autonomous deployments it is crucial to integrate recovery mechanisms that protect against the accumulation of faults, which could eventually lead to system failure. However, since fault detection processes are not infallible and may yield false positives (Lynch 1996), simply blocking robots identified as faulty from participating in the oracle is not advisable. Our approach to this problem introduces oracle *reputation tokens*, assigned based on each robot's contributions to the oracle and used as weights for its future contributions. A robot that has incurred many faults will have a lower reputation and, consequently, a smaller impact on the oracle's outcome. However, robots are always allowed to continue participating in the oracle and can therefore rebuild their reputation.

6.5.3 Reputation system

After each voting round, robots receive or lose reputation tokens based on their contributions. As mentioned above, the purpose of this is to enable the system to autonomously recover from faults: without such a measure, robots that are frequently or persistently faulty could continue to negatively affect the system performance and, as faults accumulate on different robots over time, this could lead to a system-wide failure.

The voting decision occurs when the $\frac{2}{3} K T$ quorum is reached. Given the assumption that faulty robots possess fewer than $\frac{1}{3}$ of the existing tokens, and since robots must deposit exactly $K t_i$ tokens with each report, the absolute majority of tokens in each proposal originate from non-faulty robots. This ensures that

the outcome of the voting round is correct. The robots that vote favorably to this outcome should receive reputation tokens, while the robots that vote against it should lose tokens. This reallocation of reputation tokens improves the *safety* of the protocol for subsequent rounds. Additionally, it is important to penalize robots that abstain from participating in the voting round. This can be achieved either by removing some of their reputation tokens, or by diminishing their relative reputation by issuing extra reward tokens to the robots that participated in the voting round. Doing so makes the system more efficient and less likely to come to a deadlock (i.e., improves its *liveness*), since the most active robots gain assets relative to robots that vote less often, or malicious robots that abstain from voting.

Different applications of the oracle may require different choices in the design of the reputation system or its parameters. A common downside of introducing reputation systems in peer-to-peer networks is the risk of introducing vulnerabilities that allow malicious agents to illegitimately acquire reputation tokens (Sabater and Sierra 2005).

In our experiments, the number of tokens earned by a robot for sending report r is given by the reputation gain function G:

$$G(r, \mathcal{C}_j) = \begin{cases} \frac{I_c}{|M(\mathcal{C}_j)|} + \sum\limits_{r' \in \mathcal{C}_j \setminus M(\mathcal{C}_j)} \frac{r'.d}{|M(\mathcal{C}_j)|}, & \text{if } r \in M(\mathcal{C}_j), \\ -r.d, & \text{otherwise} \end{cases} \tag{6.1}$$

where r is a report in the cluster $\mathcal{C}_j$, $M(\mathcal{C}_j)$ is the set of reports that form the (weighted) absolute majority of the cluster, I_c is the *issuance constant*, the parameter which establishes how many reputation tokens are generated each time a voting round finishes. More specifically, $M(\mathcal{C}_j)$ is defined as the following set:

$$M(\mathcal{C}_j) = \{r \in \mathcal{C}_j \mid r.v = v^*\}$$

where v^* is the unique weighted majority vote such that

$$\sum_{\substack{r \in \mathcal{C}_j \\ r.v = v^*}} r.d > \frac{1}{2} \sum_{r \in \mathcal{C}_j} r.d.$$

Equation 6.1 states that robots sending reports in $M(\mathcal{C}_j)$, in addition to receiving their deposited tokens $r.d$, will also receive an equal share of the deposits of robots sending reports not in $M(\mathcal{C}_j)$. The robots sending reports in $M(\mathcal{C}_j)$ also receive an equal share of the I_c newly issued tokens. This means that the supply of reputation tokens changes after each voting round: after the j^{th} voting round, the new quantity of circulating reputation tokens is $T_j = T_{j-1} + I_c$. Note that as more reputation tokens are added to the system, the $\frac{2}{3}$ threshold applies to the new quantity of circulating tokens, and the weights associated with existing deposits are adjusted

to match the new token supply. In our experiments, we use $I_c = \frac{T_0}{\lfloor K^{-1} \rfloor}$, where T_0 is the total amount of reputation tokens at the start of each experiment.

6.5.4 Smart contract implementation

The Swarm Oracle's fault-tolerant protocol and reputation system are implemented through Algorithm 1, deployed as an Ethereum smart contract. The smart contract provides two functions for oracle participants: `Report`, which allows them to send reports by depositing the required amount of reputation tokens, and `Query`, which returns all accepted and pending proposals.

The `Report` function is executed by sending blockchain transactions that encode the required deposit and report data. Once these transactions are added to the blockchain ledger, the state of the smart contract is updated. The `Query` function retrieves information from the current state of the smart contract, which is locally stored on each robot; this operation does not require a transaction.

The smart contract additionally employs four internal functions: a clustering function `Cluster(`$\boldsymbol{r}$`)`, a distance function between two observations `Distance(`o_1`,`o_2`)`, a function `Aggregate(`$\mathcal{C}_j$`)` to aggregate the reports in cluster $\mathcal{C}_j$ into a proposal, and a reward function `Reward(`$r, \mathcal{C}_j$`)` that implements the logic of the reputation system. We employed incremental k-means clustering and a Euclidean distance function; the proposals correspond to the geometric center of the observations, weighted by the deposited reputation tokens; and the reward function is the one given in Equation 6.1. The modular design of these functions allows for easy replacement with other options better suited to different applications of the oracle. This design choice helps keep the protocol generic.

6.6 Chapter summary

In this chapter, we introduced and addressed the blockchain oracle problem through Swarm Oracle: a decentralized hardware-oracle network in which a swarm of autonomous robots collectively senses the environment and reaches global agreements on the state of the environment.

Swarm Oracle combines sparse peer-to-peer exchanges among robots with a Byzantine fault-tolerant protocol deployed through a smart contract. Robots submit reports containing an environment observation, a vote, and a deposit of reputation tokens. Reports are clustered into proposals, and a final decision is made when a proposal accumulates a supermajority (two-thirds) of the circulating reputation tokens. This ensures that even if Byzantine robots control one-third of the tokens, they cannot force incorrect agreements.

Algorithm 1: Swarm Oracle Consensus Protocol

Input : $\mathcal{S}$, Ω
Output : $\mathcal{L} \in pow(\Omega)$
Parameters : K, I_c, q, T_0
State : $\mathcal{C}$, $\mathcal{L}$, $\boldsymbol{r}$, $\boldsymbol{t}$, T
Initialization : $\mathcal{C} \leftarrow \emptyset$, $\mathcal{L} \leftarrow \emptyset$, $\boldsymbol{r} \leftarrow [\,]$, $\boldsymbol{t} \leftarrow \left[\frac{T_0}{|\mathcal{S}|}\right]^{|\mathcal{S}|}$, $T \leftarrow T_0$,

Function `Report`$(r\{o, i, d, v\})$

- $(t, b) \leftarrow$ `Balance`(i) ;
- **if** $r \in \boldsymbol{r}$ ***or*** $r.d < Kt$ ***or*** $r.d > b$ **then**
  - **return** 0 ; ▷ `Reject report`
- $\boldsymbol{r} \leftarrow \text{append}(\boldsymbol{r}, r)$;
- $\mathcal{C} \leftarrow$ `Cluster`$(\boldsymbol{r})$;
- **foreach** $\mathcal{C}_j \in \mathcal{C}$ **do**
  - $quorum \leftarrow \sum_{r \in \mathcal{C}_j} r.d \geq qKT$; ▷ `Quorum condition`
  - **if** *quorum* **then**
    - $majority \leftarrow \exists v^*$ s.t. $\sum_{\substack{r \in \mathcal{C}_j \\ r.v = v^*}} r.d > \frac{1}{2} \sum_{r \in \mathcal{C}_j} r.d$; ▷ `Majority condition`
    - **if** *majority* **then**
      - $M(\mathcal{C}_j) \leftarrow \{r \in \mathcal{C}_j : r.v = v^*\}$;
      - **foreach** $r \in \mathcal{C}_j$ **do**
        - $\boldsymbol{t}_i \leftarrow \boldsymbol{t}_i + \mathtt{G}(r, \mathcal{C}_j)$; ▷ `Distribute rewards`
      - $\mathcal{L} \leftarrow \mathcal{L} \cup$ `Aggregate`$(M(\mathcal{C}_j))$; ▷ `Update consensus set`
      - $T \leftarrow T + I_c$; ▷ `Update token supply`
    - $\boldsymbol{r} \leftarrow [\, r \in \boldsymbol{r} \mid r \notin \mathcal{C}_j \,]$; ▷ `Update cluster set`
    - $\mathcal{C} \leftarrow \mathcal{C} \setminus \{\mathcal{C}_j\}$; ▷ `Update reports array`

Function `Query()`

- $proposals \leftarrow \{$`Aggregate`$(\mathcal{C}_j) : \mathcal{C}_j \in \mathcal{C}\}$;
- $consensus \leftarrow \mathcal{L}$;
- **return** $(proposals, consensus)$

Function `Balance`(i)

- $reputation \leftarrow \boldsymbol{t}_i$;
- $balance \leftarrow \boldsymbol{t}_i - \sum_{\substack{r \in \boldsymbol{r} \\ r.i = i}} r.d$;
- **return** $(reputation, balance)$

These trustworthy agreements are then provided to a blockchain to support decision-making. They can be used to update the state of a shared memory that the robots use for coordination; or be provided to smart contracts on public blockchains as a service.

We evaluated Swarm Oracle in a perception task where 12 robots estimate the RGB values of environment features using noisy and individually biased camera sensors. We show that, even under limited-range communication, the system withstands several types of coordinated attacks from Byzantine robots. In short-run experiments, we show that Swarm Oracle rejects proposals from Byzantine robots with manageable costs, even when one-third of the swarm is Byzantine and colludes to perform potentially devastating attacks. Long-run simulations showed Swarm Oracle's built in resilience and *self-healing*: as consecutive correct agreements are reached, attackers lose reputation tokens and the cost of achieving consensus decreases toward attack-free levels, restoring safety margins and operational efficiency.

Chapter 7

Applications: Enabling Opportunities for Robot Swarms

Building on the contributions presented in previous chapters, this chapter focuses on *applications*. It demonstrates how the methods developed throughout the thesis—including smart contracts, token-based reputation systems, and application-driven instantiations of Swarm Oracle—can enable the opportunities identified in Chapter 3. In particular, we emphasize (i) *decentralized supervision*, (ii) *data consistency* in distributed learning and swarm mapping, and (iii) *economy-based interactions* to regulate information exchange and enhance trust in realistic swarm robotic deployments.

Concretely, the objectives of this chapter are to:

1. Demonstrate *real-time decentralized supervision* of a foraging swarm using a smart contract that stores resource patches discovered by robots and assigns robots as foragers (Section 7.2);

2. Show how a blockchain can provide a *conflict-free and secure shared data structure* for decentralized federated learning, and how smart-contract-based defenses mitigate model poisoning (Section 7.3);

3. Improve the *robustness of collaborative SLAM* by performing an on-chain geometric consistency check among map pieces submitted by different robots, thereby securing the optimization of a shared map. We analyze latency as well as the security level required to resist colluding Byzantine robots (Section 7.4);

4. Explore *market-based mechanisms* to manage how social information is used for robot's navigation in the absence of GPS. Localized contracts are generated among two robots, creating a common reference frame and commitment to

a crypto tokens for social information. Blockchain-based mechanisms are then studied to promote the exchange of honest and good quality social information (Section 7.5).

The chapter begins with a review of related work on swarm foraging, federated learning, and collaborative mapping (Section 7.1). We then present four applications: smart-contract supervision for foraging (Section 7.2); blockchain-backed collective learning (Section 7.3); Byzantine-tolerant swarm mapping (Section 7.4); and market-based social navigation (Section 7.5). Finally, Section 7.6 summarizes the chapter and closes the thesis.

7.1 Introduction and related work

In this section, we review related work in three application areas: social foraging and navigation in swarm robotics (Section 7.1.1); federated learning in multi-robot systems (Section 7.1.2); and collaborative localization and mapping (Section 7.1.3). For each application area, we discuss how integrating a blockchain and the methods proposed in this thesis can potentially advance the state of the art.

In Section 3.3.2, we discussed economics-aligned robot behaviors and swarm design, reviewing work that connects robotics with methods from economic theory—providing additional relevant background for Section 7.5.

7.1.1 Social foraging and navigation in swarm robotics

Foraging is one of the most studied behaviors in swarm robotics because it models a wide range of application scenarios, including search and rescue, agriculture, mining, waste collection, and planetary exploration. It can be described as the combination of two subtasks: searching the environment for objects, and performing actions on those objects. To highlight the benefits of blockchain-based coordination, we focus on tasks based on central-place foraging (Houston and McNamara 1985, Winfield 2009), where agents locate and transport objects back to a target location (the "nest").

Typical swarm-robotics algorithms are inspired by ant colonies, where social foraging emerges through stigmergy—indirect communication mediated by the environment (Deneubourg et al. 1990, Grassé 1959). Researchers have attempted to emulate this mechanism using chemicals (Fujisawa et al. 2014, Salman et al. 2020), augmented reality (Font Llenas et al. 2018, Talamali et al. 2020), infrared signals emitted by smart environmental features (Khaliq et al. 2014, Valentini et al. 2018), and virtual pheromones implemented via robots acting as beacons (Campo et al. 2010, Hoff et al. 2013, Nouyan et al. 2009). However, these approaches either require

specialized infrastructure (e.g., smart environments or chemical sensing/depositing) or reduce efficiency by dedicating part of the swarm to beaconing. Moreover, Aswale et al. (2022) recently showed that stigmergy can be particularly fragile in the presence of malicious agents. For these reasons, explicit communication has also been studied to coordinate collective foraging, inspired by the recruitment dances of honeybees (Biesmeijer and Vries 2001, Seeley 1983).

Pitonakova et al. (2014, 2018) compare robot swarms that recruit at the nest with swarms of individualist foragers. They show that when resources are scarce or difficult to find, recruitment improves total collection; conversely, when resources are abundant, individual foraging can be preferable because it reduces both *physical interference* (collisions among robots targeting the same resources) and *informational interference* (robots being misguided by incorrect social information). Despite these insights, prior work does not provide a coordination strategy nor a methodology for validating exchanged information to limit such interference.

In this thesis, we have discussed how to secure inter-robot information exchange and protect shared data from Byzantine robots. Swarm Oracle was used in Zhao et al. (2023†) to let robots reach collective agreements on GPS coordinates of resource locations, and in Zhao et al. (2025†) to improve swarm efficiency by leveraging heterogeneous capabilities such as faster movement or lower positioning error. In these works, Swarm Oracle's reputation system flags robots that provide incorrect or low-quality social information, thereby reducing *informational interference*. However, these approaches assume access to (noisy) GPS localization, which may not be available in many deployments (e.g., underground, underwater, or remote exploration).

In this chapter, we present two fundamentally different ways to use a blockchain to improve cooperation and performance in foraging swarms. In Section 7.2, we implement a decentralized supervisor (see Section 3.3.5) that improves collective performance by allocating robots to resources scattered in the environment. The supervisor follows simple rules, established from top-down design, that reduce *physical interference* and improve performance relative to unsupervised robots. We discuss trade-offs in hardware cost, scalability, and flexibility, especially under changing environmental conditions (e.g., resource abundance and distribution).

In Section 7.5, we build on Van Calck et al. (2023†), which extends direction-based social foraging methods where robots exchange *relative directions* toward resources or the nest to construct social paths that support simultaneous transport and guidance (Ducatelle et al. 2014, Ducatelle et al. 2011, Gutiérrez et al. 2009b, Miletitch et al. 2013, Sperati et al. 2011). We show that such mechanisms are vulnerable to robots providing incorrect or low-quality information: social paths can break, forcing robots to revert to exploration, and malicious robots can exploit the system to gain a foraging advantage. To address these risks in open (trustless)

swarms, we study an information marketplace in which robots trade social information about resource locations for crypto-token rewards, and we evaluate reward designs that incentivize honest reporting and sustain robust cooperative foraging.

7.1.2 Federated learning in multi-robot systems

Federated learning (McMahan et al. 2017) is an approach to distributed machine learning that has the advantage of keeping data local and private while distributing the costs of training models among the nodes in a network. Therefore, it could hold great promise in swarm robotics applications. However, federated learning typically assumes a centralized server that aggregates local models.

Majcherczyk et al. (2021) proposed Flow-FL, which applies federated learning in a robot swarm to learn a global model of robots' motion patterns. In their system, each robot records trajectories of itself and nearby robots and trains a local deep neural network (a long short-term memory (LSTM) layer followed by a dense layer). This model can be useful to make predictions regarding peers' movements, or to coordinate collective motion. The robots' local models are then aggregated into a global model. Flow-FL uses virtual stigmergy (Pinciroli et al. 2016), a shared-memory mechanism designed to tolerate asynchronous interactions and network partitioning in swarms. However, neither virtual stigmergy nor Flow-FL provide mechanisms to manage conflicts caused by Byzantine behaviors (e.g., sending different models to different peers). In realistic swarm deployments, conflicts may arise due to unforeseen circumstances or adversarial behavior, leading to divergence of the shared model across the network.

Implementing federated learning in swarm robotics therefore raises two challenges. First, the system must synchronize a conflict-free shared data structure for the global model without introducing a single point of failure. Second, it must address the oracle problem by protecting the learning process from incorrect yet non-conflicting inputs. The first problem is naturally addressed through the use of a blockchain, while the second requires an oracle mechanism. Our results show that Flow-FL is vulnerable to a single Byzantine robot submitting random model weights, which could arise from sensor faults or from a compromised robot performing *model poisoning* attacks (Fung et al. 2020). Without defenses, a single Byzantine robot can potentially cause a complete system failure by targeting the shared model.

Securing federated learning is challenging because local models are trained on private data, making it difficult to verify model quality without access to that data (Xu et al. 2019). Robust aggregation algorithms have been proposed, including FedAvg (McMahan et al. 2017) and the Byzantine-tolerant Krum (Blanchard et al. 2017) and its variants (Bareilles et al. 2026, Colosimo and De Rango 2023). Fung et al. (2020) additionally discuss potential attack vectors and Sybil protection.

However, these algorithms are typically implemented under a centralized server model.

In Section 7.3, we construct a decentralized implementation of federated learning in a swarm using the blockchain framework presented in Chapter 4. Robots broadcast local models as blockchain transactions, and a smart contract aggregates them into a shared model. We then present defense mechanisms based on a smart contract that mitigate Byzantine behavior and enable secure learning of the shared model. We use the same experimental scenario and robot controllers as Flow-FL (Majcherczyk et al. 2021), providing a baseline for comparison.

7.1.3 Collaborative localization and mapping

Autonomous robots operating in unknown environments must perceive and interpret their surroundings to navigate effectively. In Simultaneous Localization and Mapping (SLAM), robots build a map while estimating their own spatial pose within it (Durrant-Whyte and Bailey 2006, Placed et al. 2023). Decades of research on single-robot SLAM have produced efficient methods for mapping and localization (Ayache and Faugeras 1988, Crowley 1989, Engel et al. 2014, Lourakis and Argyros 2009, Mur-Artal et al. 2015, Newcombe et al. 2011, Zhang et al. 2023).

More recently, researchers have explored collaborative SLAM (C-SLAM), where multiple robots jointly construct a shared map. C-SLAM distributes mapping across robots, offering potential for improved efficiency, better localization accuracy, and increased robustness (Lajoie et al. 2022). Early approaches adapted single-robot algorithms by combining data from many robots, using tools such as Kalman or particle filters (Fox et al. 2000, Rodriguez-Losada et al. 2004). Later work formulated C-SLAM as a graph-optimization problem solved either centrally or in a distributed manner (Nerurkar et al. 2009), yielding substantial performance gains (Grisetti et al. 2010, Saeedi et al. 2016b). Modern systems increasingly incorporate data fusion and machine learning components to improve robustness (Chen et al. 2023).

Despite progress, many C-SLAM systems remain limited to small teams and structured environments. SLAM processes are data-intensive, requiring high bandwidth and capable compute platforms. Scalability is therefore a key challenge, as computational demands increase with the number of robots and density of the collected data. Heterogeneous robots introduce additional complexity due to differing noise profiles and sensing modalities (Chang et al. 2022). Robustness also remains a challenge, facing issues such as *perceptual aliasing* where distinct locations generate similar data that robots fail to disambiguate (Tian et al. 2022). As a result, even modern frameworks (Chang et al. 2022, Cieslewski et al. 2018, Cramariuc et al. 2023, Fernandez-Cortizas et al. 2024, Huang et al. 2022, Lajoie and Beltrame 2024, Lajoie et al. 2020, Schmuck et al. 2021, Tian et al. 2022, Zhong

et al. 2024) face limitations in data management, scalability, and robustness.

Robot swarms, being composed of many cooperating robots that operate without relying on central control or external infrastructure, have the potential to provide a foundation for scalable and fault-tolerant collective mapping (Kegeleirs et al. 2019). Despite this potential, there is a methodological gap between C-SLAM and swarm robotics research due to additional challenges which arise from the lack of a central server, and the hardware and communication constraints of swarm platforms. Kegeleirs et al. (2021) provide a perspective on swarm-based SLAM and its challenges. Lajoie and Beltrame (2024) introduced Swarm-SLAM, which addresses key challenges in swarm settings by supporting intermittent and bandwidth-constrained communication. In Swarm-SLAM, each robot estimates its trajectory and map locally; then, when robots meet, they exchange compact information to align and merge their maps through an optimization process executed by delegated robots. Despite its significance, open research questions remain regarding scalability, security, and data management in Swarm-SLAM.

In particular, Lajoie et al. (2019) highlight the role of perceptual aliasing, where misclassified environment features introduce conflicting or incorrect constraints between robots' trajectories that propagate into optimization. Such misclassifications can arise from perception noise and faults or from the environment itself which may contain similar features. C-SLAM frameworks are often described as robust or resilient (Chang et al. 2022, Lajoie et al. 2020, Tian et al. 2022), typically referring to mechanisms built into the optimization algorithm to filter noise and reject outliers, mitigating perceptual aliasing (Cadena et al. 2016). However, their effectiveness can degrade under heterogeneous noise and persistent or coordinated Byzantine behavior. Without protections, Byzantine robots can inject incorrect constraints that disrupt the generation of the collective map and robots' capability to self-locate, potentially leading to deployment failure.

In Section 7.4, we propose complementary protections to secure and validate inter-robot exchanges before this information enters existing SLAM pipelines. To do so, we use a blockchain to detect and discard mapping conflicts before they enter Swarm-SLAM, improving robustness under sparse connectivity and Byzantine behavior. Concretely, robots broadcast transactions describing their noisy trajectory estimates based on odometry and observations of environment features. A feature observation is accepted only after validation by three distinct robots: a smart contract applies a geometric consistency test (the *triangle identity*) to transformed feature observations, and only mutually consistent reports are forwarded to the map optimizer.

7.2 Decentralized foraging supervisor

It has become clear that the use of blockchains or similar shared data structures can have a large potential impact on design and self-organization of swarm robotic systems. However, further research is required to understand the extent of this impact, as well as its potential drawbacks. In the previous chapters, we have studied how blockchains can help achieve consensus in the presence of Byzantine agents. However, while these secure agreements were enabled by and stored in a blockchain, they were not used to make collective action decisions during the swarm's operation.

In this section, we validate the claim that smart contracts can be valuable when applied to the real-time coordination of robot swarms. In this context, a smart contract is control code that is executed in a decentralized manner by the swarm; that is, each robot executes the code independently and the swarm comes to an agreement on its output. From a micro-level perspective, the individual robots collect local information and deliver it to the smart contract by broadcasting local messages. From a macro-level perspective, the smart contract extends the swarm's ability to self-organize by aggregating the input of the robots and returning action policies on which the robots can act in real-time.

To demonstrate this, we deploy a blockchain smart contract to act as a "decentralized supervisor" during a collective foraging task in which the swarm needs to collect resources spread in an unknown environment. The robots broadcast transactions that contain information obtained from scouting the environment for resources. This information is aggregated by the smart contract into a shared database of resource locations. The blockchain consensus protocol guarantees that these transactions are executed in an orderly, conflict-free manner, and that all robots reach an agreement on the most recent state of this database. Afterwards, the smart contract distributes the available robots (recruits) to the various resources, while (i) prioritizing resources with better quality; and (ii) limiting the number of foragers per resource. These simple rules are shown to increase the resource collection rate and energy efficiency during the task.

The rest of the section is organized as follows. In Sections 7.2.1 and 7.2.2, we introduce the foraging task, the environment, and the robots' finite-state controller. In Section 7.2.3, we describe the foraging supervisor smart contract. In Sections 7.2.4 to 7.2.6, we report on the blockchain latency and the experimental results on scalability and flexibility. In Section 7.2.7, we discuss the implications and limitations of using a blockchain to coordinate robots in real-time, and provide guidelines for future work.

7.2.1 Environment and task

The goal of the swarm is to retrieve resources from the environment and deposit them at the nest. Resources have various qualities that yield a different reward when deposited. The performance of the swarm is measured in terms of the total *reward* collected, and of the *scouting efficiency*, which is the ratio between the reward collected and the distance traveled by the robots while exploring the environment. Each experiment lasts 15 minutes.

The environment consists of a square arena with the nest located at the center. The size of the arena is a function of the number of robots (i.e., we maintain a constant robot density of 3 robots per m^2, and the nest occupies 10% of the arena's total area. The nest is divided into two areas (Figure 7.1): an external annulus, where robots can deposit resources; and an internal circle, where robots can idle. The nest broadcasts a homing signal which allows the robots to navigate to the nest from any location.

Resource *patches* are circular areas distributed randomly in an annulus centered on the nest and with radii 0.83 m and 1.44 m. *Resources* are individual items contained in a patch that the robots can collect and deposit at the nest. The patches can be of 4 different types (red, green, blue and yellow), and the resources collected from each type yield a different reward (2, 4, 6 and 8, respectively). Once a patch runs out of resources, an identical patch spawns elsewhere.

We consider three *distributions* of patches and resources in the environment. In all distributions, approximately 3% of the environment area is covered with patches, and there is an identical number of red, green, blue and yellow patches.

- **Scattered small patches (SSP):** The patches are distributed uniformly in the annulus, have a diameter of 16 cm and contain 10 resources (Figure 7.1, left).
- **Scattered big patches (SBP):** The patches are distributed uniformly in the annulus, have a diameter of 36 cm and contain 15 resources (Figure 7.1, middle).
- **Clustered small patches (CSP):** The patches are distributed according to a normal distribution that is biased towards the upper left quadrant of the arena, have a diameter of 16 cm and contain 10 resources (Figure 7.1, right).

7.2.2 Robot foraging behaviors

The robots are controlled by a *finite-state machine*. At each simulation step, the robots perform a routine corresponding to their current state, as well as a *local peer discovery* routine.

The *finite-state machine* starts at the state `Scout` and is composed of 5 states:

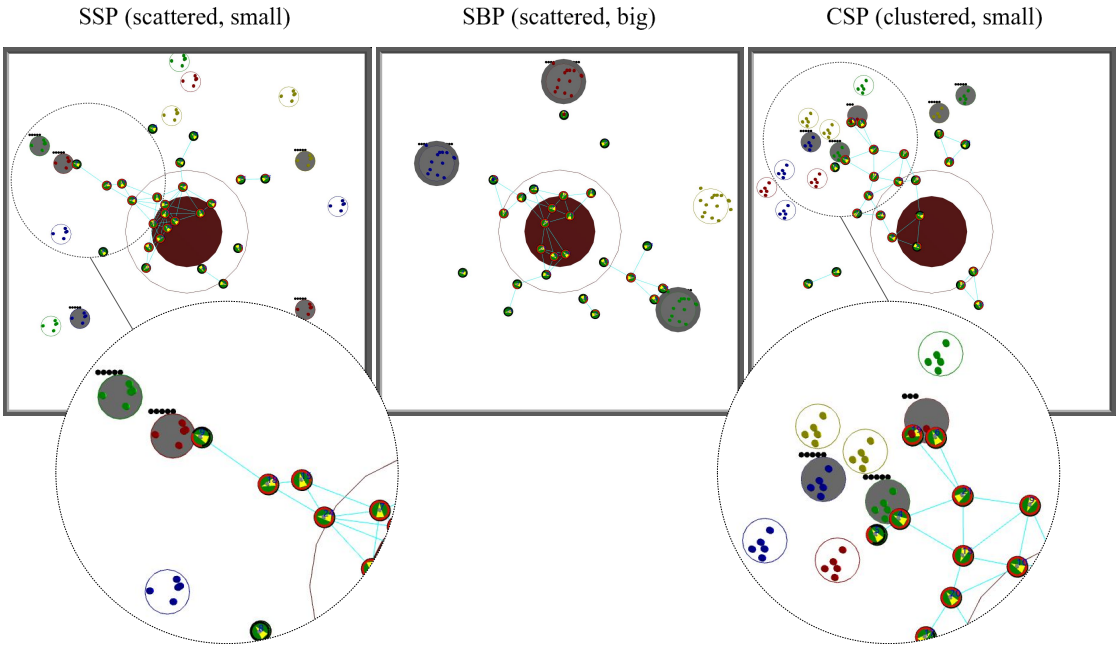


Figure 7.1: Foraging experimental environments. Frames from experimental trials for the different resource patch distribution: **SSP** (left), **SBP** (middle) and **CSP** (right). The patches are circles with items inside (the resources). A gray background means that the patch is included in the blockchain database, and the black dots above represent the remaining quantity of resources according to the blockchain database. The brown circle and annulus in the center are the nest and its deposit area, respectively.

- **`Idle`** Wait for 30 seconds; then, transition to `Scout`.
- **`Scout`** Perform a random-walk, with a duration sampled from $\mathcal{N}(\mu = 40$ s, $\sigma = 2$ s) and store the discovered patches in a list stored locally; then, broadcast a transaction to execute `update_patches(scouted_patches)`. Once the transaction is included in a block, delete the list and transition to `Plan`.
- **`Plan`** Return to the nest using the homing signal and invoke a call to `query_patches()`. If assigned to forage a resource, transition to `Search`; otherwise, broadcast a transaction to execute `assign_patch()`, and wait until it is included in a block. If the transaction fails (no resources available to forage), transition to `Idle`.
- **`Search`** Navigate from the nest towards the direction of the assigned patch and search its neighborhood for 10 seconds. If resources are

found, transition to `Forage`; otherwise broadcast a transaction to execute `update_patches(depleted_patch)` and transition to `Scout`.

- **`Forage`** Collect a resource from the patch and navigate to the nest using the homing signal. Then, broadcast a transaction to execute `update_patches(current_patch)` to inform that one resource was removed. Once the transaction is included in a block, deposit the resource and, if there were more resources, transition to `Search`; otherwise, transition to `Scout`.

In addition, the robots execute the communication protocol described in Chapter 4, with a maximum communication distance of 30 cm.[1] The larger communication range enhances connectivity and thereby the swarm's ability to synchronize the blockchain. In return, it becomes possible to reduce the block period to 2 s, reducing delays in processing information and enabling timely foraging decisions. Next, we present the logic behind the supervisor smart contract.

7.2.3 Smart contract

Our smart contract allows robots: (i) to store information regarding discovered resource patches; (ii) to enlist themselves as recruits in order to forage at a certain patch; and (iii) to query information about the known resource patches. It ensures that the highest-reward resources are prioritized for foraging, and that there is a limit on the number of foragers per patch.

The robots can interact with functions by broadcasting *transactions* (to execute the function on the blockchain network), or by invoking *calls* (to execute the function locally and read its output). Our smart contract has 3 functions:

- **`update_patches(patches[])`**: The input is a list of formatted strings which contain the relevant information about a patch: position, radius, quality, and quantity of resources. If the position is unique (within an error margin), a new patch is added to the database, otherwise an existing patch is updated.

- **`assign_patch()`**: If there are available patches (i.e., patches with fewer foragers than the maximum number allowed), then the transacting robot is assigned as a forager to the highest quality patch.

- **`query_patches()`**: Returns a database of resources, including the current foragers for each resource.

[1] In Chapter 5 the maximum communication range was 10 cm, however with a significantly higher robot density (robots per square meter in the environment).

In general, this supervisor smart contract extends the swarm's ability to self-organize, and thus improves its collective performance—without compromising the decentralized nature and desired properties of a robot swarm. In the following sections, we analyze the *latency* introduced by the blockchain-based supervisor layer, and analyze the system's *scalability* and *flexibility*.

7.2.4 Blockchain consensus latency

The *block period* (B_p) has a large effect on the information delay introduced by the blockchain: if it is too high, it reduces the possibility to employ shared knowledge to perform time-critical decisions. Conversely, if it is too low, it increases the frequency of block production which leads to (i) higher costs of communication, computation, and data storage; and (ii) an increased rate of blockchain forks which contain redundant, or more dangerously, conflicting information. As such, B_p should be adjusted to the levels of connectivity among robots' and application requirements, as discussed in Chapter 5.

Figure 7.2 (left) shows the *block propagation time*, which is the difference between the timestamp at the moment a robot receives a block and the timestamp at the moment the block was produced (in other words, the time it took for a block to be disseminated through the network from its producer to any other robot). Figure 7.2 (right) shows the *block production time*, which is the time difference of the timestamps between two consecutive blocks on the final version of the blockchain. The first metric is calculated online by the robots, while the second is calculated offline after the experiment is finished.

On the left, we see that the majority of blocks (about 70%) are received within 2 s. This observation justifies the choice of $B_p = 2\,\text{s}$, as there is a high chance that the previous block has been disseminated through the network before it is time to produce the next block. This insight is then corroborated by the plot on the right: 90% of the blocks on the final blockchain were produced within 2 s to 3 s from the previous one, meaning that the proof-of-authority preferred sealers are receiving blocks timely and consensus is not stalling significantly.

7.2.5 Scalability under increasing swarm sizes

In previous research (Pacheco et al. 2020[†]), we set the block period to 15 s. Here, we set a block period of 2 s, which naturally leads to higher storage costs since more blocks are generated. Nonetheless, empty blocks (i.e., blocks without transactions) are lightweight since most of the data is stored within the transactions. As such, the majority of the costs come from transactions, whose structure and contents depend on the application rather than on the blockchain protocol and parameter choices.

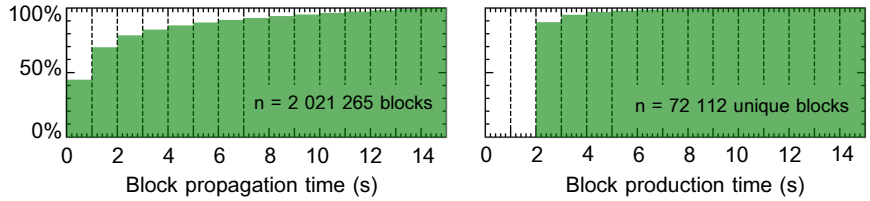


Figure 7.2: Blockchain consensus latency study. The histograms represent cumulative probability distributions, and are generated from the combined data of all experiments performed in this study. **Left:** In 70% of instances where a robot received a block, that block had been produced less than 2 s before; and in 100% of the instances, less than 15 s. **Right:** The minimum and ideal production time is equal to $B_p = 2$ s, but additional delays occur due to network delays (e.g., temporary unavailability of the preferred block producer).

Data storage Figure 7.3 (left) shows the data storage required by each robot, which is seen to increase linearly with the number of robots in the swarm. In larger swarms, more transactions are broadcast thereby increasing storage requirements. On average, each robot needs to store between 6.25–8 MB for 15 minutes of operation, depending on swarm size. The robots in our experiments are *full blockchain nodes*, i.e., each robot stores the complete blockchain history. In a real deployment this might not be necessary, since only the most recent information is relevant for the robots' operations, and the task of storing the blockchain history can be delegated to external agents when connection is available, or it can be segmented and stored by the robots in a distributed manner. In this case, the hardware-limited robots would host *light blockchain nodes* (Buterin 2014), while remaining able to verify the status of the blockchain and of the transactions by leveraging cryptographic primitives. For these reasons, we do not expect data storage to pose a scalability problem in a real deployment.

Foraging performance Figure 7.3 (right) shows that the swarm is capable of maintaining performance (the total reward collected increases with the number of robots) as the environment size, number of robots and quantity of resources scale accordingly. However, we also observe decreasing performance returns (the total reward collected increases sublinearly with the number of robots). Rather than a limitation of our blockchain-coordinated approach, this seems to occur due to the layout of the environment, which is prone to interference at the centrally located nest when the swarm size increases.

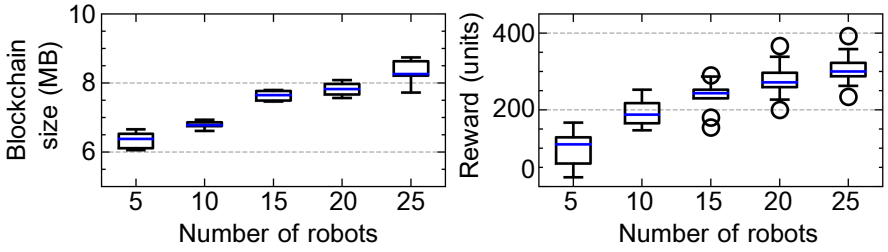


Figure 7.3: Foraging scalability results. Left: The storage space required for each robot grows linearly with the number of robots, at a rate of approximately 1 MB per 10 robots. Right: The collected reward grows sublinearly with the number of robots, due to the increasing rate of physical interference at the centrally located nest. These experiments were repeated 25 times using the SSP distribution. The duration of each trial is 15 minutes.

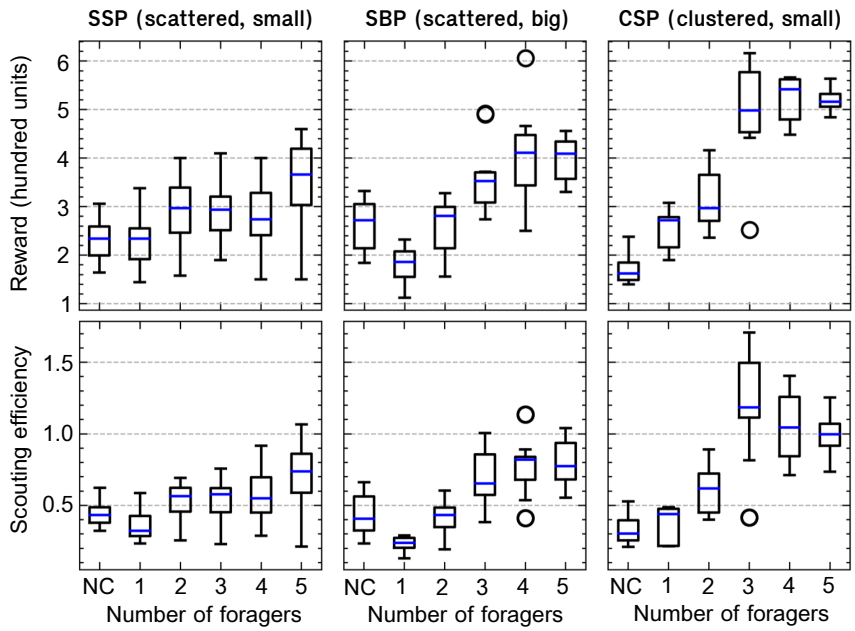


Figure 7.4: Foraging flexibility in different environments. Performance results for three distributions: SSP (left), SBP (middle) and CSP (right). The top row shows the reward collected by the swarm at the end of the experiment, and the bottom row the scouting efficiency. The x-axis (number of foragers) is a parameter in the smart contract which limits how many robots can be tasked as foragers for each resource patch. The uncoordinated robot swarm shows "NC". A swarm of 25 robots was used, and the experiments were repeated 10 times.

7.2.6 Flexibility under different resource distributions

SSP distribution In this environment there is a large number of patches randomly spread on the map. Previous research (Pitonakova et al. 2018, Wilson 2000) indicates that individualist foragers tend to perform well, or even better than cooperating robots (when the benefits of cooperation do not overcome the negative effects of interference). In Figure 7.4 (left) the total reward collected saturates at 2 foragers per patch, but is consistently higher than the non-collaborating swarm ("NC" in the x-axis). The scouting efficiency can be significantly higher but also has a high spread. This happens because cooperating robots, when lucky, will discover higher quality patches and better allocate foragers to those resources.

SBP distribution The blockchain-coordinated swarm is capable of retrieving 50% to 100% more reward, and being twice as efficient in scouting for resources, as seen in Figure 7.4 (middle). In this environment, the advantage of coordination is more pronounced since (i) the patches last longer as they contain more resources, and (ii) they are larger in size and there is therefore less interference.

CSP distribution The blockchain-coordinated swarm is capable of retrieving more than double the reward and being 2 to 5 times more efficient during scouting, as seen in Figure 7.4 (right), than non-collaborating swarms. This occurs because the scouting robots that move in the direction of the resource cluster are very successful, while other robots do not find any resources. The ability to aggregate and share information prevents unsuccessful robots from idling or wasting energy performing redundant exploration. Conversely, given the tight aggregation of resources, the foraging efficiency quickly drops as the number of recruits increases above 3 due to physical interference between robots.

7.2.7 Discussion

The blockchain enables a new form of distributed control for robot swarms that uses explicit communication and coordination, while preserving decentralization and local exchanges of information. It is important to note the contrast between the *macro perspective* that is used when creating smart contract supervisors and the *micro perspective* that is more frequent in the design of robot controllers. In our research, we present the two perspectives as complementary, since the behavior of individual robots emerges from local sensing and interactions, while the blockchain is regarded as an additional layer that is reserved for high-level decision making.

Unlike approaches based on authoritative central or hierarchical control, a blockchain-based supervisor supports decentralized top-down governance of the swarm by encoding global rules that are enforced through a horizontal decision-making process in which all robots participate equally. Since the proof-of-authority

consensus algorithm is robust to the unavailability of up to 50% of the network nodes, our blockchain-coordinated robot swarm does not have a single point-of-failure and could be deployed in situations where a system that relies on information traveling to and from supervisors would fail (for example, in environments with limited or no communication infrastructure).

In general, we showed that the coordination rules provided by a smart contract supervisor can improve the performance of the robot swarm during the foraging task, while keeping reasonable data storage costs and manageable delay in the control loop. These are positive results that showcase the potential of deploying blockchains for the real-time coordination of robot swarms in a wider range of scenarios.

Scalability, consensus latency and storage costs Two potential challenges in blockchain-based control are *consensus latency*—that is, the time required for transactions and blocks to propagate through the network and become part of the blockchain—as well as the costs of *data storage*, since each robot maintains a local copy of the blockchain database. These aspects can raise scalability concerns in terms of communication overhead and hardware requirements for robot swarms. In Sections 7.2.4 and 7.2.5, we discussed these concerns and showed that they are manageable for swarms of different sizes. The storage requirement per minute of operation for a swarm of 25 robots is 540 KB, which is significantly higher than the 23.06 KB reported for a swarm of 24 robots in Chapter 5. In these foraging experiments, each transaction contains more data, as it includes a JSON file describing the properties of environmental patches (color, value, and quantity of resources available). Reducing the block period to 2 s also increases storage costs; however, because blocks without transactions require only 508 B (see Chapter 4), this effect is minimal relative to the costs of transactions. The swarm can accommodate the shorter block period efficiently, since the robots maintain good connectivity due to their 30 cm communication range.

Flexibility In foraging, cooperation is not always an advantage (Pitonakova et al. 2018). Sharing information can lead to increased physical interference, for example, when the robots forage the same resources rather than finding a balance between exploitation and exploration. It may also lead to informational interference, which occurs when robots propagate incorrect or outdated information (e.g., if a resource patch becomes depleted during the time the information is being processed, or if the robots' sensors provide inaccurate positions). Our smart contract supervisor improved the performance of the swarm (in terms of the *reward* collected and the *scouting efficiency*) by aggregating information about resource patches from the robot scouts, and assigning resource patches to robot recruits—thus minimizing the impact from both forms of interference. In Section 7.2.6 we reported the performance results of a blockchain-coordinated robot swarm and compared

them to those obtained with a swarm of uncoordinated robots, which explore the environment and forage resources as they discover them individually. Our results show that uncoordinated swarms perform relatively better in environments with many scattered small patches, since the resources in patches become exhausted faster, leading to information interference due to the list of patches in the smart contract contained exhausted patches more often.

In future work, the coordination logic in the smart contract could be improved to further exploit the shared knowledge it contains. For example, theory from optimal foraging (Stephens and Krebs 1986) can be applied in order to improve the energetic balance of the swarm; or the smart contract can incorporate a machine learning algorithm which assigns resources in order to maximize a reward function. A smart contract with the ability to learn and adapt to the environment can significantly improve flexibility, by adjusting its rules according to the observed performance or information reported by the robots during online operations.

7.3 Securing collective learning

In this section, we explore the use of blockchain technology to enable a decentralized implementation of federated learning in a robot swarm. To achieve this, each robot trains a local model using the data it collects. Rather than relying on a central server, the swarm orders and aggregates these local models through a blockchain-based smart contract. Prior work has used virtual stigmergy as a distributed data structure to host the shared model (Majcherczyk et al. 2021, Pinciroli and Beltrame 2016); however, virtual stigmergy is an optimistic protocol that neither provides security against the broad range of faults that may arise in real-world peer-to-peer deployments nor offers mechanisms for decentralized conflict resolution. To address these limitations, we use the Ethereum blockchain (Buterin 2014), using the methods presented in Chapter 4.

Although the blockchain provides secure aggregation and state replication via smart contracts, the federated learning process must still be protected against malfunctioning or malicious robots that submit incorrect or low-quality local models—otherwise known as model poisoning attacks. Blockchain therefore serves a dual purpose in our approach: it acts as (i) a distributed computation platform that maintains and synchronizes the aggregated model, and (ii) a trusted substrate for implementing security mechanisms that mitigate the risks posed by Byzantine robots.

Warnat-Herresthal et al. (2021) introduced blockchain-based federated learning in the context of medical data to improve privacy and security. Here, we investigate the specific challenges posed by networks of mobile robots, which operate under local communication constraints. For these reasons, the network topology changes

rapidly, causing information to reach different robots at different times. This poses a challenge for federated learning, since to train new model iterations, robots require up-to-date versions of the aggregated model.

The rest of this section is organized as follows. In Sections 7.3.1 to 7.3.3, we present the experimental setup and methodology, including the environment and task, the robots and Byzantine behaviors, and the smart contract that hosts the aggregation and security mechanisms. In Sections 7.3.4 to 7.3.7, we report experimental results, starting with a baseline study without security and then evaluating robustness under different Byzantine behaviors. In Section 7.3.8, we discuss the implications, limitations, and potential future directions for secure collective learning via blockchain-based federated learning in robot swarms.

7.3.1 Environment and task

A swarm of 15 robots navigates a $5 \times 5\,\mathrm{m}^2$ arena while avoiding obstacles and other robots. Obstacles—five cylinders with radius 0.15 m and five boxes with base $0.3 \times 0.3\,\mathrm{m}^2$—are distributed uniformly at random in the arena. Each experiment lasts 5 000 s. A visual representation of the arena is shown in Figure 7.5.

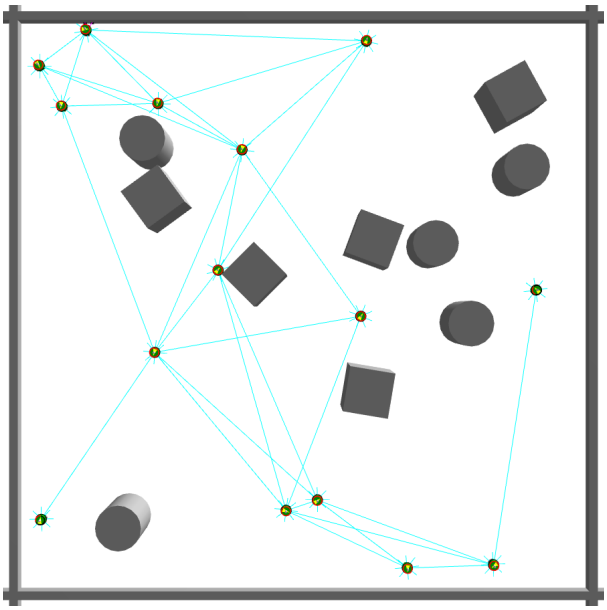


Figure 7.5: Experimental arena. The Pi-puck robots (small green circles) are connected by cyan lines when they can record each other's trajectories through their range-and-bearing sensors. The short cyan rays around each robot represent collision-avoidance sensors used to detect walls (delimiting the arena), obstacles (gray cubes and cylinders), and other robots.

Following Chapter 4, we deploy Pi-puck robots in ARGoS (Pinciroli et al. 2012) and run Ethereum (Buterin 2014) nodes in Docker (Merkel 2014) containers. In this setup, robot controllers send transactions and interact with smart contracts, while blockchain synchronization and maintenance are executed by the Docker containers in the background. The source code for the setup and all experiments is available online (Pacheco et al. 2024b†). We evaluate performance using the following metrics.

Average loss We define average loss as the weighted average of the validation loss of each robot model participating in a given aggregation round.

Tokens gained Tokens gained is the number of tokens accumulated by robots at the end of an experiment through the reward system. Depending on the experiment, we analyze the difference between the tokens gained by Byzantine and non-Byzantine robots.

7.3.2 Robot and Byzantine behaviors

The Pi-pucks are equipped with an array of infrared sensors for obstacle detection, a range-and-bearing board to estimate the motion and identities of nearby robots, and a Wi-Fi module to synchronize blockchain blocks and transactions. All communication occurs within a maximum range of 2.5 m.

Each robot moves in the environment while avoiding other robots and obstacles, and records data about its own motion and the motion of its peers using the range-and-bearing sensor. A motion trajectory is a sequence of (x, y) positions over 10 s. If the connection with another robot is interrupted during recording, the corresponding trajectory is discarded.

Every 100 s, a robot enters a training phase. During this phase, it retrieves the latest aggregated model from the blockchain and trains it on the most recently collected data using TensorFlow (Abadi et al. 2016). We use the same hyperparameters as in (Majcherczyk et al. 2021): a batch size of 20, a learning rate of 0.001, and 20 epochs.

After local training, the robot includes its updated model in a blockchain transaction and broadcasts it so it can contribute to the next update of the shared model. The robot then resumes data collection.

(A) Byzantine behaviors

We consider three types of Byzantine robots: faulty, malicious, and smart.

– *Faulty* robots submit random model weights drawn uniformly from $[-0.5, 0.5]$. In real deployments, such behavior could occur if a robot's sensors are obstructed or damaged and therefore yield arbitrary values.

– *Malicious* robots submit, as their "trained" model, the shared model from the previous aggregation round retrieved from the blockchain. Such behavior could occur if a robot is compromised and the attacker attempts to bypass the outlier-rejection threshold and slow down model training.

– *Smart* Byzantine robots attempt to predict a plausible update to the shared model without performing training or data collection. This allows them to obtain tokens without incurring the associated costs. Specifically, a smart robot z estimates the average magnitude of weight changes during the last aggregation step (from $t-1$ to t) and applies a random variation of comparable magnitude to the current model weights $w(t)$. The weights submitted by robot z are:

$$w_{z,i}(t+1) = w_i(t) + 2r \cdot \Delta(w(t), w(t-1)) \,, \tag{7.1}$$

$$\text{where } \Delta(w(t), w(t-1)) = \frac{1}{N} \sum_{i=1}^{N} |w_i(t) - w_i(t-1)| \,,$$

and r is a uniformly distributed random value in $[0, 1]$, regenerated independently for each weight $w_{z,i}(t+1)$, with $i \in \{1, 2, \ldots, 2848\}$ in the robot's weight vector w_z.

7.3.3 Model aggregation and security mechanisms

In the federated learning literature, security mechanisms are typically studied in centralized settings, where a server is responsible for protecting the shared model against poisoning attacks and low-quality updates. In our approach, aggregation and security mechanisms are implemented through a smart contract.

(A) Aggregation through a smart contract

Local-model aggregation is performed by a smart contract exposing two functions to the robots:

- `submitModel`: robots upload locally trained models, which are aggregated into the shared model using FedAvg (McMahan et al. 2017). The shared model is updated whenever a specified fraction of robots—a *quorum*—have submitted models. We set the quorum size to $\lfloor q \times N_r \rfloor$ with $q = 50\%$ and N_r the swarm size, yielding a quorum of 7 out of 15 robots. If the quorum is too large, learning can slow down when robots fail to submit (e.g., due to disconnections). If it is too small, security degrades because Byzantine robots may constitute a majority within an aggregation round.

- `getModel`: robots retrieve the latest shared model for local training. If a robot has not synchronized to the latest blockchain state due to network partitioning, it may train from an outdated model. This can be mitigated by appropriate parameter choices and a consensus protocol suited to the network dynamics and robot capabilities. Given the robots' speed and communication range, we use a block period of 10 s. By waiting at least 10 s between blocks, we allow enough time for robots to synchronize recent blocks and thus the latest smart contract state.

The shared model is calculated using a weighted average—based on the number of samples that the robots used to train their model—of the local models submitted by the robots in that aggregation round's quorum. In this way, models that were trained using more data have a larger impact on the shared model. This adaptation of FedAvg (McMahan et al. 2017) is shown in Algorithm 2.

Algorithm 2: ***FedAvg*** with $N_r = 15$ total robots; $q = 50\%$ quorum of robots participating; $B = 20$ local mini-batch size; $E = 20$ local epochs; $\mathcal{P}_k$ data of robot k, R aggregation rounds, loss function ℓ (mean square error), and learning rate $\eta = 0.001$.

Init: model weights $w(0)$;

Smart contract executes:;
for $t \leftarrow 1$ **to** R **do**
  $p \leftarrow \max(\lfloor q \cdot N_r \rfloor, 1)$;
  $S_t \leftarrow$ (random set of p robots);
  $N_s \leftarrow 0$;
  foreach *robot* $k \in S_t$ **do**
    $n_k(t+1) \leftarrow$ (number of samples of robot k);
    $N_s \leftarrow N_s + n_k(t+1)$;
    $w_k(t+1) \leftarrow$ `RobotTrain`$(k, w(t))$;
  $w(t+1) \leftarrow \frac{1}{N_s} \sum_{k=1}^{p} n_k(t+1) \cdot w_k(t+1)$;

Procedure `RobotTrain`(k, w)**:**
  $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B);
  for $e \leftarrow 1$ **to** E **do**
    foreach *batch* $b \in \mathcal{B}$ **do**
      $w \leftarrow w - \eta \nabla \ell(b; w)$;
  return w;

(B) Security mechanisms

Sybil and double-count protection To protect against Sybil attacks or attempts to double-count their models, robots must transfer 5 crypto tokens (scarce units maintained on-chain) to submit a model to the smart contract. Robots start with 21 tokens; if their balance falls below 5 tokens, they can no longer participate in federated learning. Tokens therefore act as participation credentials allocated through a blockchain-based reputation system that evaluates the quality of submitted models.

Outlier rejection Many defenses in federated learning begin with outlier rejection, which can prevent divergence when individual learners submit extreme updates. We use a static threshold applied to the average absolute distance between submitted model weights w_s and the model weights w_a from the last aggregation round:

$$\frac{1}{N}\sum_{i=1}^{N}|w_{s,i}-w_{a,i}|\leq 0.05\,, \tag{7.2}$$

where the number of trainable weights is $N = 2\,848$. In a pilot study, we observed that the average distance between submitted and shared models was approximately 0.01; we therefore set the threshold to five times this value. If a submission exceeds the threshold, it is excluded from aggregation and the submitting robot forfeits the tokens paid for submission.

Ranking system The ranking system sorts submitted models by their distance to the shared model. The number of ranked models in each aggregation round equals the quorum size (7 in our case). The model whose distance corresponds to the median of all submitted distances is ranked first. Subsequent ranks are assigned based on the absolute difference between a model's distance and the first-ranked model's distance.

The tokens paid alongside submissions are collected into a reward pool that is redistributed after the shared model is updated. Robots with higher-ranked submissions receive larger rewards than lower-ranked submissions, according to a reward-weight vector k. We use $k = [1, 1, 1, 1, 1, -1, -1]$, meaning that the two worst-ranked submissions are penalized (losing tokens) and the remaining five are rewarded. In addition, we scale rewards proportionally to the number of data samples used to train each model, because training on more samples typically improves performance but also increases data-collection and training costs. To do so, we first partition robots into a rewarded group R and a penalized group P

(robots in P receive fewer than 5 tokens back). Each robot i then receives:

$$R_i = \begin{cases} \left(1 + \frac{k_i \cdot s_i}{\sum_{j \in \mathsf{R}} k_j \cdot s_j}\right) \cdot 5\,\text{tokens} & \text{if } i \in \mathsf{R} \\ \left(1 - \frac{k_i \cdot s_i}{\sum_{j \in \mathsf{P}} k_j \cdot s_j}\right) \cdot 5\,\text{tokens} & \text{if } i \in \mathsf{P}, \end{cases}$$

where k_i and s_i denote robot i's reward weight and number of samples, respectively. This reward function ensures the following properties:

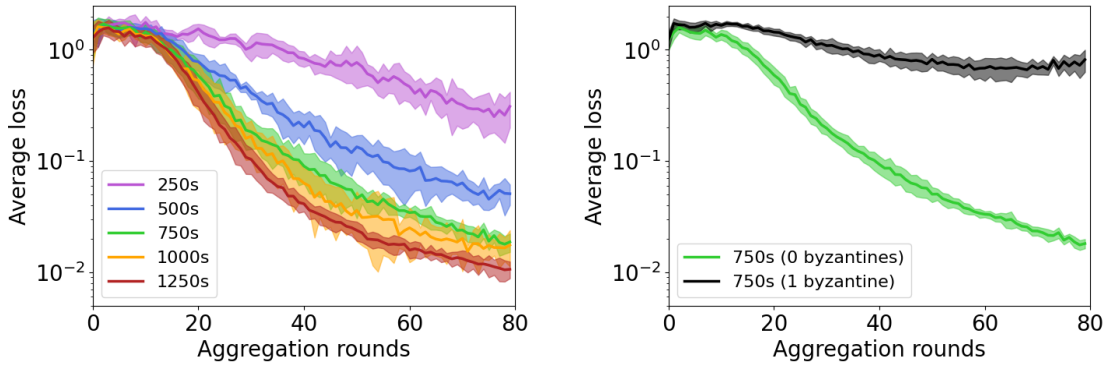
1. High rank and many samples yield high reward.
2. High rank and few samples yield low reward.
3. Low rank and few samples yield low penalization.
4. Low rank and many samples yield high penalization.

7.3.4 Baseline study without security

Storing large amounts of data on hardware-limited robots is costly, so it is important to allow old data to expire and to rely on more recent data, provided model quality does not degrade significantly. To assess a suitable *data expiration time* that meets a good performance-cost tradeoff, we vary its value from 250 s to 1 250 s in experiments with 15 non-Byzantine robots. This experiment additionally serves as a baseline for later experiments and for comparison with Flow-FL.

Figure 7.6a shows that increasing the data expiration time beyond 750 s yields only marginal improvement. Therefore, we select 750 s as the data expiration time for subsequent experiments. Comparing our results with those reported for the original Flow-FL framework (Majcherczyk et al. 2021), the convergence behavior is qualitatively similar but quantitatively different: for larger expiration times, we reach a comparable final loss of 10^{-2}, but convergence is slower. This discrepancy may be due to parameter differences, as it was not possible to retrieve all values used in the original Flow-FL study (Majcherczyk et al. 2021).

After selecting an appropriate data expiration time and confirming that our system behaves similarly to Flow-FL (despite the blockchain integration), we test resilience to the introduction of a single *faulty* Byzantine robot, still without security mechanisms. Figure 7.6b shows that the system is vulnerable to even one faulty Byzantine robot, motivating the need for security mechanisms. In the remaining experiments, we use the smart contract implementing the security mechanisms described above. We conduct 20 runs for each configuration, ranging from 0 to 7 Byzantine robots.



(a) The effect of data expiration time on the average loss with no Byzantines.

(b) The effect of introducing one Byzantine robot without security mechanisms.

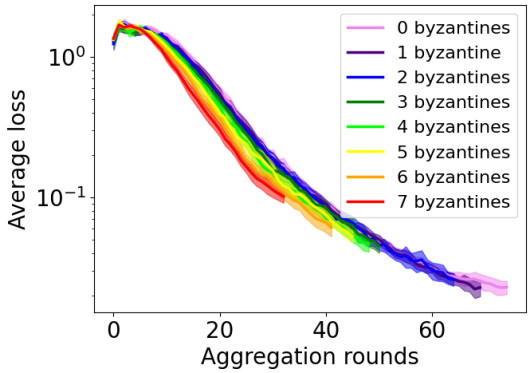
Figure 7.6: Average loss without security mechanisms. **(a)** Increasing the data expiration time leads to reduced average loss since more data is used for learning. Expiration times above 750 s show similar loss curves. Therefore we use 750 s for all subsequent experiments. **(b)** A single Byzantine robot is sufficient to prevent model convergence (with a data expiration time of 750 s). Shaded areas depict 95 % confidence intervals, with 5 trials for (a) and 10 trials for (b).

7.3.5 Rejecting outliers from faulty robots

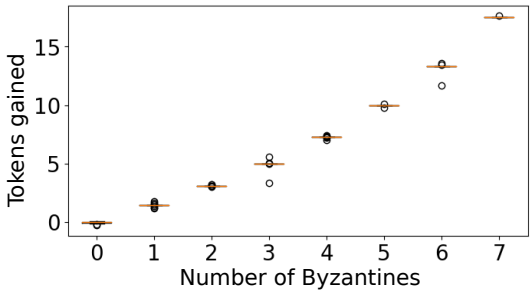
We first introduce faulty Byzantine robots. Since faulty robots submit random weights, we expect outlier rejection to exclude most of their submissions. These experiments therefore primarily evaluate the effectiveness of outlier rejection against non-adversarial faulty behavior. Figure 7.7a shows that the security mechanisms effectively reject inputs from faulty Byzantine robots, and that average loss is largely insensitive to the number of Byzantine robots. Because the smart contract requires 5 tokens per submission and redistributes these tokens among robots whose models are accepted, we expect non-Byzantine robots to gain tokens on average while Byzantine robots lose tokens. This behavior is confirmed in Figure 7.7b.

7.3.6 Ranking system to detect malicious robots

Although outlier rejection protects against outlier models, a malicious Byzantine robot can still influence model training by submitting slightly perturbed models that remain within the threshold. Individually, such submissions may have limited impact; however, colluding malicious robots could manipulate the shared model. To mitigate this, we introduced a ranking mechanism that orders submissions according to their deviation from the median model weights across all submissions and assigns token rewards based on rank.



(a) Average loss for different numbers of faulty Byzantine robots.



(b) Tokens gained by non-Byzantine robots.

Figure 7.7: Average loss and tokens gained by faulty robots. **(a)** As the number of Byzantine robots increases, fewer aggregations occur within the fixed 5 000 s experiment duration. Rejecting outlier models improves robustness, but fewer aggregation rounds are completed because fewer robots submit reliable models while the quorum remains fixed (7 robots). **(b)** As the number of Byzantine robots increases, non-Byzantine robots gain more tokens. Shaded areas depict 95 % confidence intervals over 20 trials.

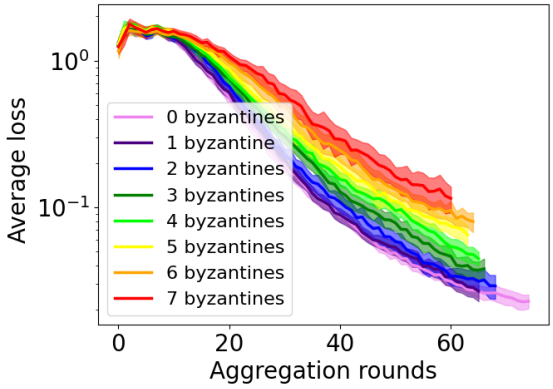
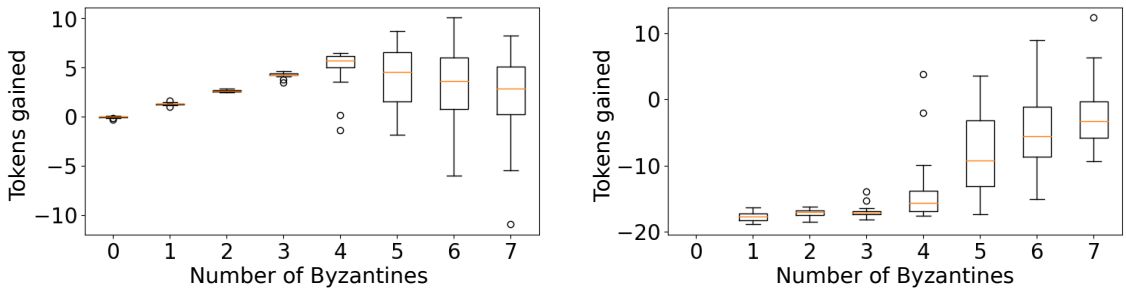


Figure 7.8: Average loss with malicious robots. Average loss for different numbers of malicious Byzantine robots. Because many malicious submissions remain within the outlier threshold, they are often included in aggregation. As a result, we do not observe the sharp reduction in aggregation rounds seen with faulty robots. Although the ranking mechanism preserves convergence, Byzantine robots are able to slow down the training of the shared model.



(a) Tokens gained by non-Byzantine robots

(b) Tokens gained by Byzantine robots

Figure 7.9: Tokens gained by malicious robots. With up to 3 malicious Byzantine robots, the ranking mechanism rewards non-Byzantine robots and penalizes Byzantine robots. With 4 Byzantine robots, this is no longer guaranteed; with 5 or more, Byzantine robots are likely to gain tokens.

Figure 7.8 shows that the ranking mechanism preserves convergence even in the presence of malicious Byzantine robots. However, as the number of Byzantine robots increases, convergence slows and average loss increases. Unlike faulty robots, whose submissions are almost always rejected, malicious robots' submissions are often accepted, which slows learning while also allowing aggregation rounds to occur more frequently. In the fixed-duration experiments (5 000 s), this yields more aggregation rounds under malicious behavior than under faulty behavior (compare Figure 7.7a and Figure 7.8). Figure 7.9 shows that non-Byzantine robots gain tokens up to 3 Byzantine robots. Beyond this point, the ranking mechanism can fail because Byzantine robots may become a majority within a quorum of 7, allowing them to maintain or gain tokens.

7.3.7 Vulnerability to smart Byzantine robots

In real deployments, robots may temporarily behave faultily and later recover. For this reason, the token-reward system is designed not only to penalize Byzantine behavior but also to reward robots that submit useful models. This keeps tokens circulating in the swarm; otherwise, tokens could be depleted and the system would halt. However, this design introduces a known vulnerability of reputation-based systems (Sabater and Sierra 2005): a smart Byzantine robot can accumulate tokens by behaving honestly initially, and later use those tokens to manipulate the shared model. We demonstrate this by running experiments with smart Byzantine robots that attempt to obtain high ranks by submitting models that follow the current trend of training.

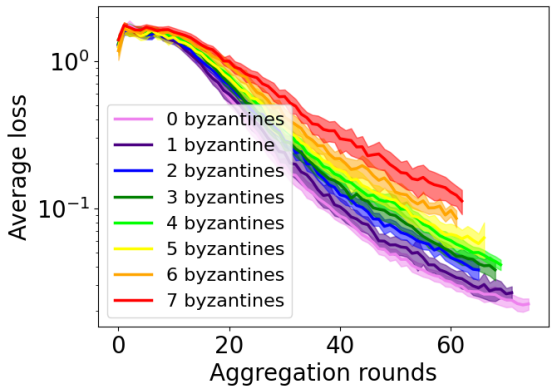
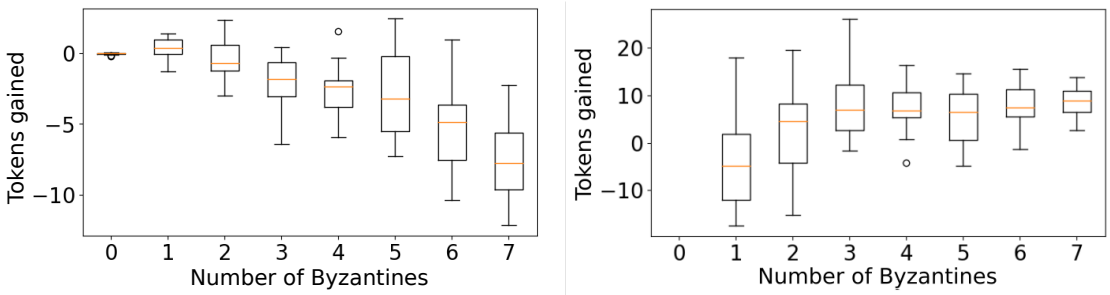


Figure 7.10: Average loss with smart Byzantine robots. The convergence speed and number of aggregation rounds are similar to those in Figure 7.8.



(a) Tokens gained by non-Byzantine robots

(b) Tokens gained by Byzantine robots

Figure 7.11: Tokens gained by smart Byzantine robots. Smart Byzantine robots consistently gain more tokens than non-Byzantine robots, especially when there are 3 or more Byzantine robots.

The loss curves in Figure 7.10 follow the same trend as in Figure 7.8: training slows as the number of smart Byzantine robots increases. With up to 4 smart Byzantine robots, the number of aggregation rounds does not decrease substantially. With more smart Byzantine robots, however, non-Byzantine robots may begin to lose participation tokens, reducing the number of effective learners and slowing aggregation. This dynamic can shift control of aggregation toward the smart Byzantine robots. This effect is visible in Figure 7.11, where smart Byzantine robots gain more tokens than non-Byzantine robots. Although the impact on convergence is not pronounced in Figure 7.10, once smart Byzantine robots control a majority of tokens, they could switch behavior and collude to manipulate the shared model.

7.3.8 Discussion

Federated learning is a promising approach for distributed systems, but its application to swarm robotics is hindered by the lack of infrastructure for secure aggregation of local models. Blockchain provides one possible substrate for implementing federated learning while preserving swarm autonomy, decentralization and fault-tolerance.

Security mechanisms Our experiments show that Byzantine robots pose a significant threat to federated learning by hindering, and potentially manipulating, the training of the shared model. We therefore implement security mechanisms that allow learning to remain effective despite Byzantine behavior. The mechanisms can provide Sybil resistance by limiting the number of outlier submissions a robot can make by using participation tokens. However, because robots may naturally experience faults over time, they risk eventually exhausting their tokens even if faults are temporary. To enable autonomous recovery from temporary faults (i.e., without requiring a technician or administrator to replenish tokens), we introduce a reputation system that both penalizes poor submissions and rewards good submissions according to how they rank each time a quorum of submissions is reached and the model is updated. Our results also show that this scheme is vulnerable to smart Byzantine robots that seek to accumulate tokens without doing useful work. Addressing this vulnerability is an important direction for future work.

Computational costs We also consider storage and bandwidth costs. Aggregation via blockchain requires robots to broadcast transactions containing model parameters, which can be several KB—significantly less than the training data; but still a non-negligible size for swarm robotics applications. Since each node stores all transactions in blocks, every submitted model is recorded on-chain. Empirically, storage cost grows approximately linearly over time (the blockchain size increases by about 16.7 KB per model submission) and reaches roughly 100 MB after 5 000 s when training a network with 2 848 weights. In this sense, the storage requirements are manageable for Pi-pucks (16 GB with a standard SD card). Nevertheless, exchanging and storing large models from all peers may be prohibitive for less capable robots.

This work constitutes an early step toward decentralized federated learning in swarm robotics, alongside Majcherczyk et al. (2021), Na et al. (2023), Xianjia et al. (2021), and Zhou et al. (2023). Combining federated learning with blockchain methodologies has the potential to enable collective learning without compromising swarm autonomy, fault tolerance, decentralization, and scalability. Future work should address smart Byzantine robots by integrating privacy-preserving methods on the submitted models and enhance data management and cost reduction, for example, through off-chain data exchanges and blockchain pruning.

7.4 Byzantine-tolerant swarm mapping

In this section, we address key limitations and security vulnerabilities of the Swarm-SLAM framework (Lajoie and Beltrame 2024) using blockchain technology. In Swarm-SLAM, robots build a collective map by aggregating individual robots' maps through pose graph optimization (PGO). However, Swarm-SLAM provides no mechanism to guarantee that the swarm reaches an agreement on a single, collective map, particularly in the presence of faulty robots. In fact, PGO is typically executed using ad hoc rules (e.g., delegating optimization to the robot with the lowest ID).

We address these limitations by integrating a blockchain layer that provides (i) a consistent ordering of robots' contributions and (ii) an explicit coordination mechanism for PGO without resorting to heuristic leader selection. Moreover, we show that Swarm-SLAM is vulnerable to Byzantine robots. To mitigate this threat, we deploy a smart contract that validates and enforces geometric consistency constraints on inter-robot loop closures, thereby protecting the convergence of the collective map.

This section is organized as follows. In Section 7.4.1, we review Swarm-SLAM and analyze its vulnerabilities. In Section 7.4.2, we detail the experimental setup. In Section 7.4.3, we present our smart-contract-based geometric validation of inter-robot loop closures. In Section 7.4.4, we introduce a security-level parameter to increase robustness to Byzantine collusions. In Section 7.4.5, we describe the token-based reputation scheme we implemented. Finally, in Sections 7.4.6 and 7.4.7, we report robustness and resilience results, and in Section 7.4.8, we discuss the proposed approach and outline future directions.

7.4.1 Swarm-SLAM and its vulnerabilities

Swarm-SLAM, proposed by Lajoie and Beltrame (2024), is a state-of-the-art C-SLAM framework targeting decentralized swarms of robots with limited capabilities. It addresses key challenges by accommodating heterogeneous sensors and by operating under intermittent, bandwidth-limited communication without reliance on external infrastructure.

Swarm-SLAM consists of two modules: a *front-end* and a *back-end*.

The **front-end** interfaces with the robots' sensors to process odometry and perception data and to estimate robot poses relative to environmental features. It computes descriptors for identified features, allowing robots to recognize previously observed locations and enabling the formation of *loop closures* that connect different points along their trajectories. In C-SLAM, loop closures may be intra-robot or inter-robot—when the linked poses belong to trajectories of different robots. In

the **back-end**, each robot incrementally builds a *local pose graph* whose nodes represent estimated poses and whose edges encode constraints from odometry and loop closures. When robots come into communication range, they exchange portions of their local graphs, which are then merged into a shared graph via pose graph optimization (PGO). Although the overall process is distributed in the sense that any robot can receive information from neighbors and perform PGO, existing practical implementations designate a single robot (often the lowest-ID robot) to execute PGO (Lajoie and Beltrame 2024, Lajoie et al. 2022). Swarm-SLAM does not explicitly specify a leader-election or consensus protocol for selecting that role.

Loop closures (LCs) are critical to PGO because they introduce constraints that tie together poses corresponding to the same physical location. In Swarm-SLAM, inter-robot LCs are derived from sensor data (e.g., images) exchanged when robot A recognizes a feature that robot B had previously observed. When they meet, B shares an image of the feature, which A uses to compute a geometric loop closure: a relative transformation between B's pose and A's pose, which is then incorporated as a constraint between their pose graphs.

Vulnerabilities Swarm-SLAM can be compromised by Byzantine robots in three ways:

(i) Excessive sensor noise, yielding low-quality trajectories and pose estimates that drift away from reality;

(ii) Incorrect LCs due to perceptual aliasing or erroneous estimation of the relative transformation between two poses;

(iii) Incorrect map merging via PGO: if PGO is centralized on a single designated robot, this creates a single point of failure.

While any of these errors can occur during normal operation, they can also result from malicious tampering, in which case many existing approaches based on filtering and outlier rejection may be insufficient. Research on Byzantine fault-tolerance in C-SLAM, and in Swarm-SLAM in particular, remains limited. In particular, we identified two main sources of error for LCs between robots A and B:

(a) A computes an incorrect LC transformation despite receiving a correct image from B;

(b) B sends a tampered (or otherwise incorrect) image, while A computes the LC correctly based on the received image.

Incorrect LCs introduce constraints in the PGO that do not correspond to reality, thereby disrupting optimization of the shared map (see Figure 7.13 in the following section). Figure 7.12 illustrates the different methods by which Byzantine robots can disrupt Swarm-SLAM.

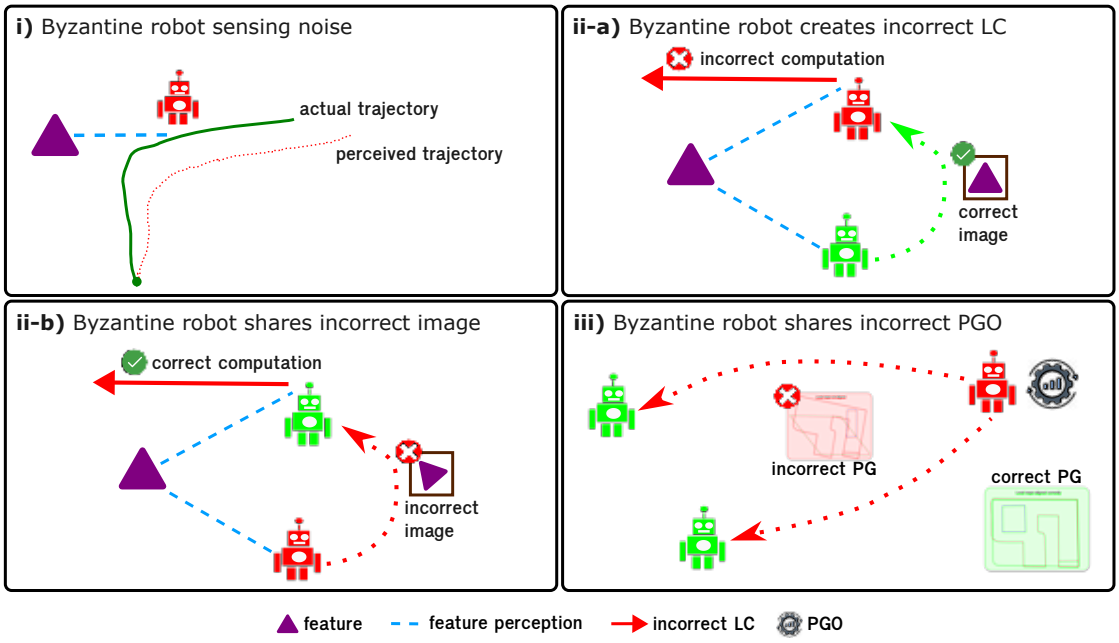


Figure 7.12: Swarm-SLAM vulnerabilities. **(i)** Byzantine robots (red robots) with excessive sensor noise estimate inaccurate trajectories, which may disrupt PGO and therefore the collective map. **(ii-a)** The Byzantine robot receives a correct image from a peer but generates an incorrect LC. **(ii-b)** The Byzantine robot sends a tampered image, prompting the peer to compute an incorrect transformation. **(iii)** When a single Byzantine robot performs PGO, it can tamper with the optimization results. Image adapted from Moroncelli et al. (2024[†]).

7.4.2 Environment and task

We evaluate our method in 10 simulation trials, each lasting 40 minutes. The experimental setup is distinct from that in Chapter 4. The experiments were conducted using the Gazebo (Koenig and Howard 2004) simulator with a swarm of eight TurtleBot3 Waffle robots (Amsters and Slaets 2020), and each robot maintained a Toychain blockchain node (Pacheco et al. 2024c†).

Toychain is a Python research blockchain for swarm-robotics experimentation. It implements a proof-of-authority protocol and has been validated through a comparison study with the previous Ethereum setup (Pacheco et al. 2024c†). Our Swarm-SLAM protection mechanisms are implemented via a Python smart contract. The robots move and observe a 20×22 m^2 environment containing multiple wall segments and nine features used for detecting LCs (see Figure 7.13a). Robots move in straight lines and make random turns when an obstacle is detected within 0.5 m. Inter-robot communication is limited to a 5 m range.

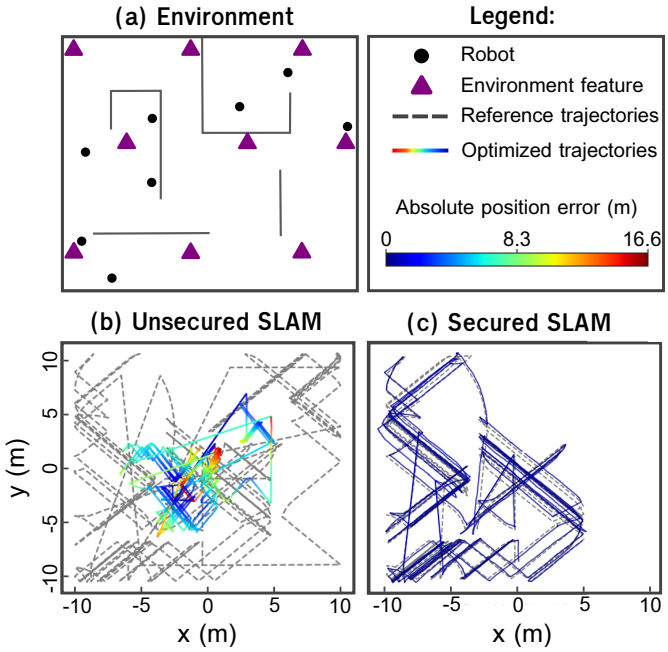


Figure 7.13: Environment in Gazebo and generated maps. **(a)** Gazebo simulation environment (20×22 m^2) with nine loop-closure features (purple triangles). Walls are shown in black; the eight TurtleBot3 robots are marked as black dots. **(b–c)** Representative trials with three Byzantine robots injecting loop closures with a constant 10 m bias. **(b)** In the unprotected case, the PGO diverges, generating a map with high error. **(c)** Using our smart contract to validate loop closures, the optimized trajectories yield approximately zero error. Image adapted from Moroncelli et al. (2024†).

7.4.3 Geometric validation of loop closures

Verifying LC correctness is non-trivial. Existing protection mechanisms can detect and mitigate perceptual aliasing to some extent; however, they cannot identify Byzantine robots because the source of an error cannot be attributed with certainty (e.g., whether it corresponds to case (ii-a) or (ii-b)).

Our solution is a smart contract that coordinates front-end data exchange among robots, as shown in Figure 7.14. When an LC is proposed, it creates a pending proposal that requires validation by three distinct robots to form a triangle of LCs. Only LCs that form a valid triangle, according to the *triangle identity*, are incorporated into PGO. While this additional step introduces latency, we show that it dramatically improves map accuracy in the presence of Byzantine robots.

To minimize data storage and enhance scalability, LCs are computed *off-chain*: robots exchange images and perform the computation locally. Afterwards, the robots delete local data and broadcast the computed LCs as transactions. Since LCs are lightweight geometric transformations, they can be stored on-chain until they are confirmed as part of a valid triangle and admitted to PGO.

A validation triangle requires three LCs from three distinct robots, arranged cyclically such that each robot acts as sender and receiver exactly once (see Figure 7.14d). Each time a new LC is submitted as a blockchain transaction, the smart contract checks whether it completes a triangle that satisfies the triangle identity constraint:

$$|l_x^i + l_x^j + l_x^k| < \epsilon \quad \text{and} \quad |l_y^i + l_y^j + l_y^k| < \epsilon,$$

where l_x^i and l_y^i denote the x- and y-components of the LC contributed by robot i, and ϵ is a sensitivity threshold.

The parameter ϵ bounds the allowable mismatch between the start and end of the composed transformations. Residuals above ϵ are treated as evidence of an invalid LC, whereas residuals below ϵ are attributed to noise and the triangle is accepted. The choice of ϵ should be matched to the target application.

Although our design protects the system against a single Byzantine robot, two Byzantine robots could collude to force acceptance of an incorrect triangle. To harden the system against collusion, we introduce a *security level* parameter.

7.4.4 Increasing security against collusion

The *security level* of an inter-robot LC counts how many times it has been independently confirmed by distinct robot triplets. A newly submitted LC starts at security level 0. Once it participates in a valid triangle with two other LCs provided by two different robots, its security level becomes 1. Each additional valid triangle that involves a new pair of robots increments this level by one.

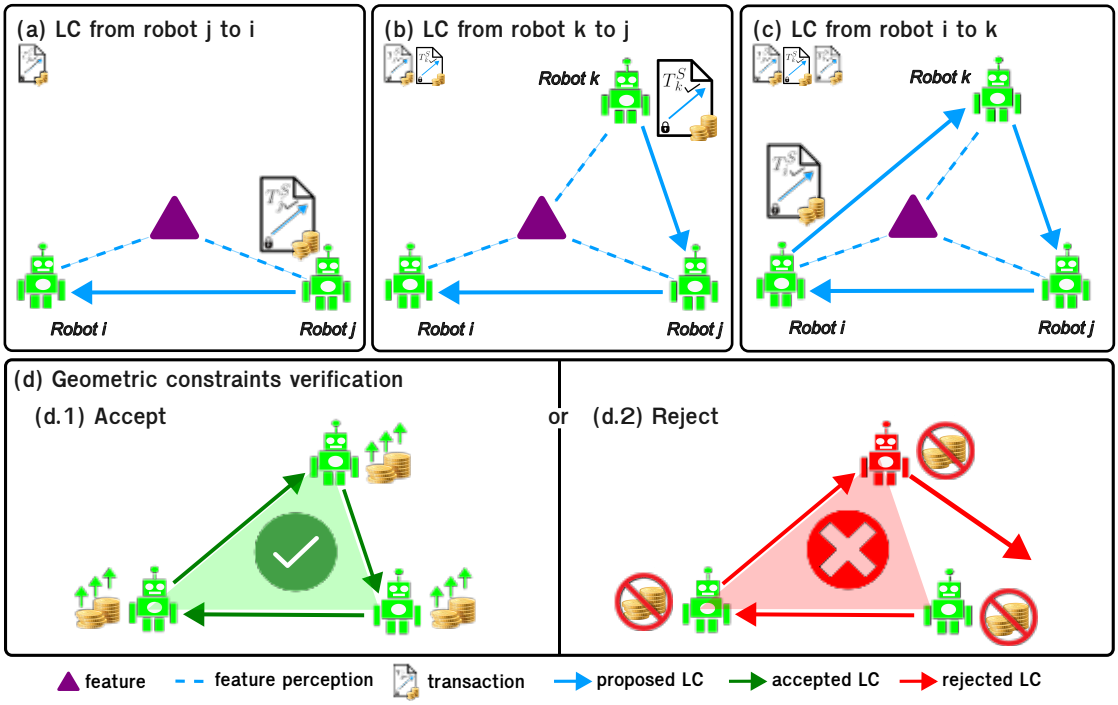


Figure 7.14: Geometric validation of loop closures. The figure illustrates triangle-based validation of LCs. **(a–c)** Each robot submits a blockchain transaction that includes the proposed LC (blue arrows) for the triangle feature (purple) and a crypto-token deposit. **(d)** The smart contract can yield two outcomes: in **(d.1)**, the triangle identity holds, so the LCs are accepted and the robots recover their authorization tokens plus a reward token; in **(d.2)**, the triangle is not geometrically valid, so the three robots lose their deposited tokens since the Byzantine robot cannot be uniquely identified. Image adapted from Moroncelli et al. (2024[†]).

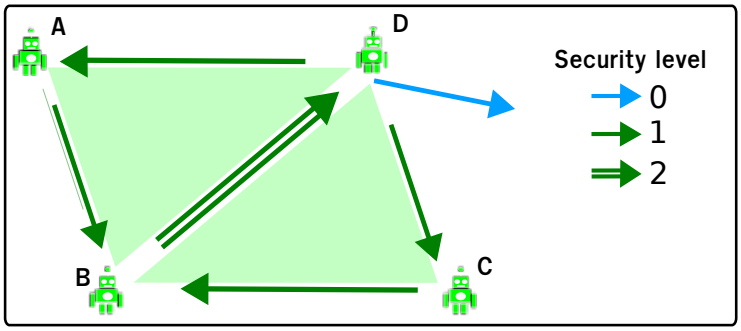


Figure 7.15: Loop closure security level. An inter-robot LC (center) reaches security level 2 after being validated in two distinct triangles formed with different robot triplets; the remaining LCs have security level 1. The blue LC is not part of a triangle so it has security level 0. Image adapted from Moroncelli et al. (2024†).

In our experiments, LCs are admitted to PGO as soon as they reach security level 1, at which point the smart contract refunds the robot's deposited authorization tokens. This choice protects against isolated Byzantine robots but does not prevent collusion. Increasing the minimum required security level improves robustness against colluding attackers at the expense of increased validation latency. Whenever an LC contributes to a security-level increase, the smart contract also awards *reputation tokens* to the involved robots. Consequently, robots that consistently submit correct LCs accumulate reputation as their results are repeatedly validated by peers. The reputation system is presented in the following section.

7.4.5 Reputation system

Beyond filtering incorrect LCs, our smart contract mitigates Byzantine behavior through an on-chain reputation scheme backed by scarce crypto tokens. Each robot is initialized with a token balance in its blockchain account and must spend tokens to participate in Swarm-SLAM. We use two token types: authorization tokens and reputation tokens.

Authorization tokens are posted as deposits alongside LC proposals (Figure 7.14a–c). The smart contract escrows each deposit and refunds it only after the corresponding LC is validated (Figure 7.14d). This mechanism limits the number of concurrently pending LCs per robot, preventing a Byzantine robot from spamming proposals indefinitely. It also limits Sybil attacks by requiring scarce tokens for participation and validation.

Reputation tokens are minted by the smart contract whenever a set of three LCs is geometrically validated. Each robot contributing one of the validated LCs receives one reputation token, which is permanently recorded on-chain. Robots

that fail to accrue reputation over extended periods can therefore be flagged as potential Byzantine robots.

7.4.6 Robustness to incorrect loop closures

We evaluate the robustness of the secured Swarm-SLAM by increasing the number of Byzantine robots from 0 to 5 (in a swarm of 8 robots total). Byzantine robots modify their proposed LCs by adding a *constant* value of 10 m to each vector component of an otherwise correct LC. Robustness is measured using the following metrics: (i) the absolute position error (APE), defined as the Euclidean distance between each point in the robots' trajectories and the ground truth; and (ii) the root mean square error (RMSE) of these distances.

Figure 7.16 shows that, in the default Swarm-SLAM algorithm without our protection layer, trajectory error increases sharply once Byzantine behavior is introduced, and worsens as additional Byzantine robots are added. Using our smart contract, by contrast, the trajectories remain close to the ground truth even with five Byzantine robots. Residual error persists due to odometry noise; this effect is reduced when more non-Byzantine robots contribute consistent measurements. Figure 7.13b–c shows examples of generated maps and accumulated APE across all robot trajectories when three Byzantine robots are present, both in the protected and unprotected cases.

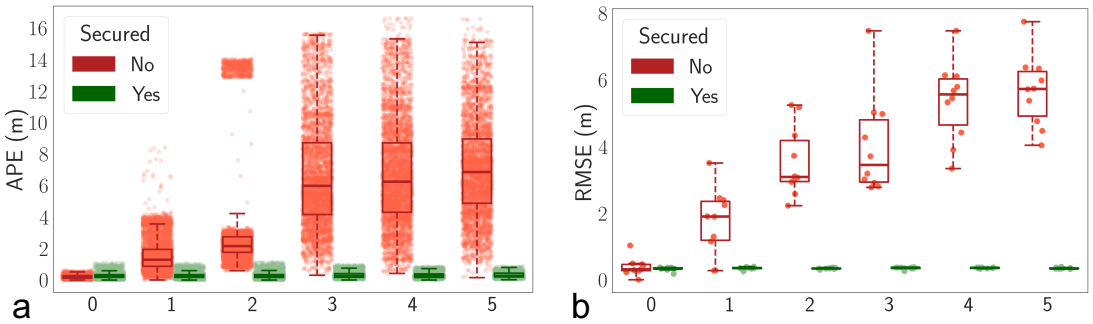


Figure 7.16: Trajectory error as the number of Byzantine robots increases. Results are shown for both secured and unsecured Swarm-SLAM. **(a)** Boxplots of the absolute position error (APE) for each point in the optimized robots' trajectories during a representative run, indicating that as the number of Byzantine robots increases, optimized trajectories deviate significantly from the ground truth. **(b)** root mean square error (RMSE) of the robots' trajectories averaged across experimental trials. We performed 10 trials with identical environment settings but different robots' initial positions. Image adapted from Moroncelli et al. (2024†).

7.4.7 Resilience and token economy

We also analyze how reputation tokens are distributed between Byzantine and non-Byzantine robots. As in previous chapters, reputation tokens serve as a proxy measure of robot reliability and can be used to improve long-term system resilience. Figure 7.17c shows that the final token balances clearly separate the two groups: Byzantine robots consistently end with zero tokens because, in our experiments, they repeatedly submitted incorrect loop closures that were never validated. Figure 7.17d highlights the performance cost of the security layer. Even in the absence of Byzantine robots, the PGO processes fewer than half of the generated loop closures, since it only consumes loop closures validated by the smart contract. This creates latency between loop-closure creation and its incorporation into PGO. As the fraction of Byzantine robots increases, the number of validated loop closures decreases further: fewer honest robots contribute correct data, and forming valid loop-closure triplets becomes markedly slower. Finally, the inset reports the fraction of loop closures reaching a security level of 1, 2, or 3. In an 8-robot swarm, only about 15% of loop closures reach security level 2 after 40 minutes, suggesting that strong protection against colluding Byzantine robots can be costly at the system scale studied here. Overall, our approach is best suited to larger swarms with many unreliable or noisy robots, where parallel validation can be exploited to increase the loop-closure validation rate while providing robustness.

7.4.8 Discussion

Swarm-SLAM (Lajoie and Beltrame 2024) is a promising framework for decentralized, collaborative mapping in robot swarms operating in unknown environments. It accommodates heterogeneous platforms and enables data aggregation through local exchanges among robots. However, realistic swarm deployments must contend with substantial uncertainty and unreliability. Inexpensive robots often have high-noise odometry and limited onboard perception and compute capabilities, which can lead to misinterpretations of the environment. Deploying such complex algorithms in swarms therefore introduces potential vulnerabilities that the robots may be ill-equipped to handle.

Our analysis shows that Swarm-SLAM is highly vulnerable to the presence of even a single Byzantine robot that shares incorrect loop closures to interfere with the PGO process. To address this, we introduce a security layer based on geometric loop-closure validation, implemented and enforced through a blockchain smart contract. This design provides clear and deterministic criteria to ensure that only validated constraints are incorporated into the optimization.

However, the proposed approach is not suitable for all SLAM applications. In small teams of highly capable robots, the additional coordination required for

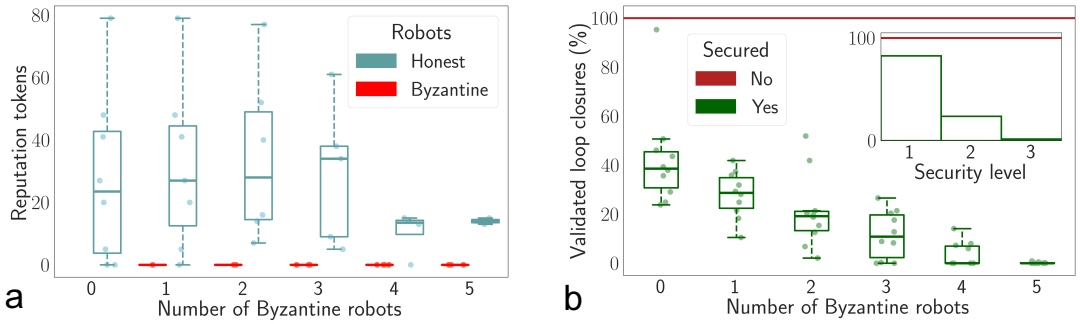


Figure 7.17: Token-based reputation and validated loop closures. Results are shown for our secured version of Swarm-SLAM. **(a)** Final reputation token balances of Byzantine and non-Byzantine robots at the end of one experimental trial. Byzantine robots remain at zero tokens because their proposed loop closures were never validated. **(b)** Proportion of proposed loop closures that are validated by the smart contract and therefore used by the PGO (10 simulation trials per condition). In unsecured SLAM, 100% of the LCs would have been accepted (red line). The inset in **(b)** reports, in the absence of Byzantine robots, the fraction of validated loop closures that reach security levels 1, 2, or 3 (figure generated from one separate experimental trial). Image adapted from Moroncelli et al. (2024[†]).

mutual validation may impose an unnecessary performance penalty. By contrast, the method is well matched to large swarms: it distributes the responsibility for certifying loop closures across multiple robots, effectively shifting trust from a single robot to a validating triplet, and thereby improving robustness to erroneous or malicious inputs.

A similar principle could be applied to the PGO process itself. Using a blockchain, one could select a rotating committee of robots to execute PGO independently and publish the resulting state. To ensure consistency across committee members, robots could commit data to versioned snapshots on-chain, defining an agreed-upon input state for each optimization round. We leave the design and evaluation of such a committee-based PGO mechanism to future work.

Future research should also consider collusion among Byzantine robots. One mitigation strategy is to exploit large swarm sizes to require higher security levels for loop-closure acceptance (i.e., more independent validations), although this comes at a significant cost. Another direction is to develop alternative geometric validation schemes that do not rely on triangles, for example higher-order polygonal consistency checks that may offer stronger resilience under coordinated adversarial behavior.

7.5 Market-based social navigation

We propose an *information market* to promote trusted exchanges among robots. The blockchain provides the ledger for economic exchanges, and a smart contract encodes the rules for social exchanges among robots. In other words, the blockchain provides the necessary decentralized substrate to implement socio-economic mechanisms that regulate interactions among robots. This has the potential to bridge the gap between the design of swarm robotics systems and methodologies found in mechanism design and game theory (Bergemann and Morris 2005, Haeringer 2018).

The key advantage of using socio-economic mechanisms to regulate agents' individual behaviors to achieve desired macro-level behaviors is the ability to leverage the extensive literature in microeconomics (Shoham and Leyton-Brown 2008). This approach can equip swarms with built-in resilience to non-cooperative robot behaviors, as well as novel methodologies for generating collective behavior.

In contrast with the work in previous chapters, in which robots contribute information to a shared memory, information exchanges here are localized: the information received by a robot is only relevant to itself, and only for a limited period of time. This locality introduces a challenge: how can robots protect themselves from malicious peers that provide misleading information? Our results show that Byzantine robots that mislead others are able to severely disrupt the system and, additionally, forage more resources than their peers. As such, this model of social foraging has built-in incentives for malicious behavior.

We discuss two forms of protection: (i) one occurs at the level of the individual, where robots wait for information from multiple peers to make informed decisions—similarly to studies on robust swarm robotics consensus where peers are stubborn, contrarian, or individualist (Masi et al. 2021, Prasetyo et al. 2018, Reina et al. 2023)—; (ii) the other is a system-wide protection mechanism in which rules for social exchange are programmed into a smart contract.

In Section 7.5.1 we describe the social navigation environment and the social odometry mechanism used by the robots. In Section 7.5.2 we show that naive information sharing is highly vulnerable to misinformation and quantify the resulting drop in foraging performance in the presence of Byzantine robots. In Section 7.5.3 we introduce an individual-level baseline defense based on skepticism and discuss its robustness–performance trade-off. In Section 7.5.4 we present a market-based systemic protection mechanism implemented via blockchain smart contracts, and evaluate three progressively stronger designs—reward sharing, outlier penalization, and outlier penalization with staking. Finally, in Section 7.5.5 we analyze how accumulated wealth relates to robot quality and conclude with a discussion of implications and limitations for open, permissionless swarms.

7.5.1 Social navigation environment

In this section, we use a different experimental setup from the one presented in Chapter 4. We use a particle-based simulator programmed in Python, which integrates a per-agent database that acts as a simulated blockchain. This simulator is structured to enable easy integration with a Python blockchain, such as Toychain (Pacheco et al. 2024c†). The robots are 2D particles that move between nest and food sites, as shown in Figure 7.18. Robots must initially explore the environment to find the locations of these sites. When robots arrive at either the nest or the food, they establish a ground-truth position by resetting their odometry variables. Afterwards, they leave to find the other site using continuously updated odometry-based estimates. Therefore, each robot maintains two vectors in local memory: one pointing to the food, and one to the nest.

After finding the food site, robots retrieve a food item and attempt to navigate back to the nest—however, this is not a trivial task for simple robots due to odometry drift, which accumulates as the robots move through the environment. In many cases, robots drift away from their target location and must return to an exploration state, wasting time and energy. Each robot has a different level of odometry sensor quality. This means that some robots are able to estimate their positions more accurately than others. To improve collective efficiency, we implement a social odometry algorithm (Ducatelle et al. 2011, Gutiérrez et al. 2010, 2009b, Miletitch et al. 2013, Sperati et al. 2011), through which robots exchange their local estimates (the vectors towards the sites) and combine them to generate a socially averaged estimate of higher quality than individual estimates.

Each directional vector is a three-tuple: `(x,y,age)`. The *age* attribute measures the time since the robot last arrived at a site and reset its odometry variables. Older estimates typically have lower quality, since they are more likely to have accumulated odometry drift.

Through our social odometry mechanism, robots exchange vectors with physical neighbors, and when a robot receives a vector with an age lower than its own, it merges the two values. We use an age-weighted averaging scheme based on Miletitch et al. (2013). In simulation, we observe that under this strategy, the swarm is able to quickly converge towards a common path that connects the two sites in an efficient straight line, as shown in Figure 7.18.

Navigation table Each robot maintains a navigation table with one entry per site (food and nest; see Table 7.1). Each entry stores: (i) the site type, (ii) the robot's current estimate of the direction vector towards the site (i.e., the position of the site relative to the robot), (iii) the age since the last odometry update, and (iv) a validity flag, which becomes false after an unsuccessful trip if the robot reaches the estimated position but does not find the site. At each time step, robots update their navigation using their odometry readings and increase the information

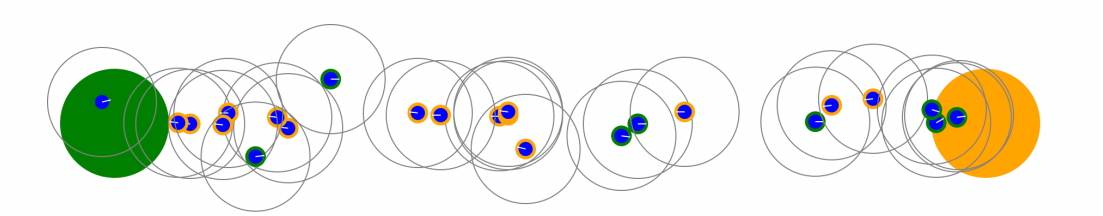


Figure 7.18: Path formation through social navigation. The figure depicts a simulation frame, where the food and nest sites are the green and yellow circles, respectively. We simulate a swarm of 25 robots (blue circles) that navigate between the two sites using social odometry based on localized exchanges of information (the gray circles show the robots' communication ranges). The swarm quickly converges towards the shortest path: a straight line connecting food and nest. The color of the robots' outlines indicates their last visited site (i.e., robots with a green outline have visited the food and are returning to the nest). Image adapted from Van Calck et al. (2023†).

age by one. If both sites have a false validity flag, the robot switches to random exploration until it finds a site or receives information from peers.

Site type	Direction vector	Age	Valid information
food	$(200, -5)$	78	`True`
nest	$(-1000, 35)$	400	`True`

Table 7.1: Example of the navigation table maintained by the robots.

7.5.2 Naive robots are vulnerable to misinformation

We call robots that trust the information provided by their peers *naive robots*. A naive robot accepts information if its age is lower than that of its own information. It then averages the information as follows:

$$\vec{\mathbf{x}} = \frac{a_{buyer}}{a_{buyer} + a_{seller}} \vec{\mathbf{x}}_{seller} + \frac{a_{seller}}{a_{buyer} + a_{seller}} \vec{\mathbf{x}}_{buyer} \,, \tag{7.3}$$

where a is the age and $\vec{\mathbf{x}}$ is the direction vector towards the nest or food. Since $a_{buyer} > a_{seller}$, Equation (7.3) places higher weight on the most recent information.

A swarm that consists of naive robots suffers drastic drops in performance when a single Byzantine robot is present. The Byzantine robot shares its direction vector $\vec{\mathbf{x}}$ rotated by 90 degrees, attempting to steer naive robots away from the socially formed path.

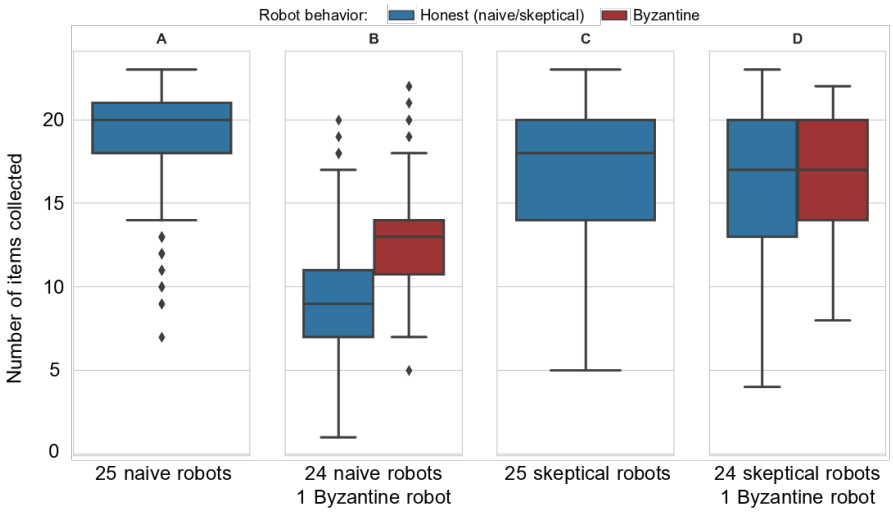


Figure 7.19: Performance of naive and skeptical robot swarms. The boxplots show the resources collected per robot. Honest robots (naive and skeptical) are shown in blue, while Byzantine robots are shown in red. **(A-B)** The performance of the naive swarm drops significantly when a single Byzantine robot is present; while the Byzantine robot itself, on average, collects more resources than its peers. **(C-D)** Swarms composed of skeptical robots are more resilient; however, they suffer from a slight performance drop compared to swarms that consist entirely of naive robots. These results are generated from 128 experimental trials lasting 15 000 simulation steps each. Image adapted from Van Calck et al. (2023†).

Figure 7.19A shows nominal performance without Byzantine robots. In a swarm composed exclusively of naive robots, each robot is able to collect 20 food units on average (i.e., each robot performs 20 return trips between the two locations). The outlier data points, below 15 units collected, correspond to the robots with the highest odometry noise values, which can sometimes fall out of the socially established path and lose access to social feedback from peers. As a result, foraging efficiency degrades as robots become lost and are forced to enter an exploration mode. Figure 7.19B shows a drop in performance of more than 50% when a single Byzantine robot is present. The results additionally show that being Byzantine offers a competitive advantage: although the swarm as a whole loses efficiency, Byzantine robots are able to retrieve more resources than the rest of the robots.

7.5.3 Individual protection through skepticism

To counter Byzantine robots, robots can become more wary of social information. *Skeptical* robots do not trust information coming from a single peer; instead,

they wait until a second peer corroborates the received information. In a given experimental trial, when honest robots implement skepticism, so do the Byzantine robots.

Skeptical robots behave like naive robots, accepting only more recent information, but integrate social information differently. Instead of immediately updating their direction vectors, they store the information in a local pending table and compare it against (i) the robot's current entry for the same site and (ii) other pending entries. For any pair (i, j) of pending entries, the *difference score* is

$$\text{diff}(i, j) = \frac{\|\vec{\mathbf{x}}_i - \vec{\mathbf{x}}_j\|}{\|\vec{\mathbf{x}}_i\|}, \tag{7.4}$$

where $\vec{\mathbf{x}}_i$ and $\vec{\mathbf{x}}_j$ are the pairwise direction vectors. We define a parameter λ that regulates the level of skepticism: if $\text{diff}(i, j) < \lambda$, the robot updates its current direction vector using Equation (7.3). In our experiments, $\lambda = 0.25$.

Figure 7.19 shows results for a swarm of skeptical robots. The increased robustness provided by individual skepticism comes with a performance trade-off. This mechanism introduces delays in updating the navigation table with the most recent social odometry, reducing individual and collective performance. Indeed, a swarm composed of 25 skeptics has lower performance than the naive counterpart (comparing plots A and C). The interquartile range also becomes wider, indicating that skeptical robots tend to become lost more often than naive robots. This is because skeptical robots with high noise can quickly fall out of the path and become lost before receiving enough information to update their direction.

Figure 7.19D shows more promising results. A swarm composed of 24 skeptics and one Byzantine maintains performance levels comparable to case C, and significantly higher than case B, where the swarm of naive robots quickly loses performance to a single Byzantine. Notably, the performance of Byzantine robots is similar to that of skeptics, meaning that there is no immediate incentive for profit-seeking robots to adhere to this particular Byzantine behavior.

Overall, skepticism is a simple measure that provides some baseline protection. As the number of Byzantine robots increases, they begin to reinforce each other's misleading information. As done throughout this thesis, we now improve security at the collective level through systemic protection mechanisms using a blockchain and smart contracts.

7.5.4 Market-based systemic protection

We propose an *information market* to promote trusted socio-economic exchanges among robots. The blockchain provides the ledger for economic exchanges, and smart contracts encode the rules for social interaction among robots exchanging information.

Robots may now receive income from two sources: (i) by foraging themselves, i.e., completing successful round trips; and (ii) by selling useful information to peers.

Robots are assumed to be self-interested and seek to maximize their income. From a robotics design perspective, accumulated wealth can be interpreted as a measure of robots' performance, either in foraging themselves or in providing information that is useful to their peers. From an application-oriented standpoint, this assumption is also reasonable, since robots may be profit-seeking agents that act on behalf of their developers or seek to achieve economic independence (Castelló Ferrer et al. 2023).

Such self-interested robots may be reluctant to help other robots, which are seen as potential competitors. For example, in Figure 7.19B, the Byzantine robot strategy is viable since, on average, it yields an economic edge over its peers—despite reducing collective performance by over 50%.

In the following sections, we discuss and explore economic mechanisms to favor cooperation and honesty in swarms of self-interested robots. We present three mechanisms tested in simulation: *Reward sharing*, *Outlier penalization*, and *Staking*.

(A) Reward sharing

We found *reward sharing* to be an effective mechanism that promotes the selling of honest and good-quality information. In this mechanism, the seller receives a share of the buyer's reward once an item is deposited back at the nest. Therefore, the foraging success of the buyer directly contributes to the seller's rewards. Otherwise, if payments were executed immediately at the moment of sale, Byzantine robots could provide false information and still be rewarded.

In our implementation, each time a successful foraging round trip finishes, the smart contract grants 50% of the reward to the successful forager, and the remaining 50% is distributed among every robot that contributed information during that round trip. However, one problem with this approach is that it only functions when the swarm is able to maintain some baseline performance level, where social information is the key factor behind swarm cohesion and efficient navigation. In that case, we would expect Byzantine robots to achieve lower payouts as they receive fewer reward shares and, additionally, must share their own rewards in order to benefit from social information themselves.

In practice, reward sharing did not prevent Byzantine robots from driving the system into a chaotic state transition[2], where the swarm's convergence towards an efficient linear path breaks down. In this case, social information becomes less useful since most robots are in exploration mode. This motivates augmenting

[2]See Kuckling et al. (2024), who discuss phase transitions between efficient and chaotic states.

reward sharing with mechanisms that explicitly penalize Byzantine robots: *outlier penalization* and *staking*.

(B) Outlier penalization

To implement reward sharing, a record of all social-information exchanges must be kept for each round trip. This record is managed by a smart contract that is updated through transactions signed by both buyer and seller. Therefore, the contract stores the set of sellers whose transactions are eligible for rewards. Exploiting this, we compare sellers' reported directions and *penalize outliers*.

To do so, each transaction stores the site type (`nest` or `food`) and the shared direction vector expressed in the buyer's coordinate system. When robot A deposits an item in the nest, the smart contract issues a reward $R = 1$. Robot A receives $R/2$, and the remaining $R/2$ is distributed among the sellers.

Let T denote the set of transactions from sellers recorded during robot A's last round trip. Each transaction $t_i \in T$ yields its sender a reward $w_i \frac{R}{2}$, where the weight $w_i \in [0, 1]$ is proportional to the number of transactions that are similar to t_i:

$$w_i = \frac{s_i}{\sum_{k=1}^{|T|} s_k}, \quad \text{where} \quad s_i = \sum_{j=1}^{|T|} \text{similar}(t_i, t_j)\,, \tag{7.5}$$

with

$$\text{similar}(t_i, t_j) = \begin{cases} 1, & \text{if site}(t_i) = \text{site}(t_j) \text{ and } |\theta(t_i) - \theta(t_j)| < \Theta, \\ 0, & \text{otherwise.} \end{cases} \tag{7.6}$$

Here, site(t) denotes the site type associated with transaction t, and $\theta(t)$ denotes the orientation of its reported direction vector. We set $\Theta = 30°$. Intuitively, s_i counts how many other transactions corroborate t_i (same site type and similar direction). The normalization in Equation (7.5) ensures $\sum_i w_i = 1$. Since reports from Byzantine robots tend to have orientations that differ substantially from the majority, they match fewer transactions and thus obtain smaller weights and lower rewards.

The impact of introducing outlier penalization on the swarm's wealth distribution is shown in Figure 7.20. In a swarm of 25 robots, each robot is allocated $\frac{1}{25} = 4\%$ of the total wealth. Therefore, any deviation from this value indicates, on average across trials, either a loss or a gain of reputation tokens. A single Byzantine robot achieves, on average, 3% of the total wealth, but as the number of Byzantine robots increases, the difference becomes less pronounced. Skeptical robots gain wealth; however, the high number of outliers and non-negligible whiskers indicate that redistribution of wealth occurs among honest robots (we analyze the correlation between odometry drift and wealth acquired below). The bottom plot shows

that Byzantine robots tend to lose a large amount of tokens at the beginning of the experiments, but then stabilize while maintaining a significant fraction of the available wealth.

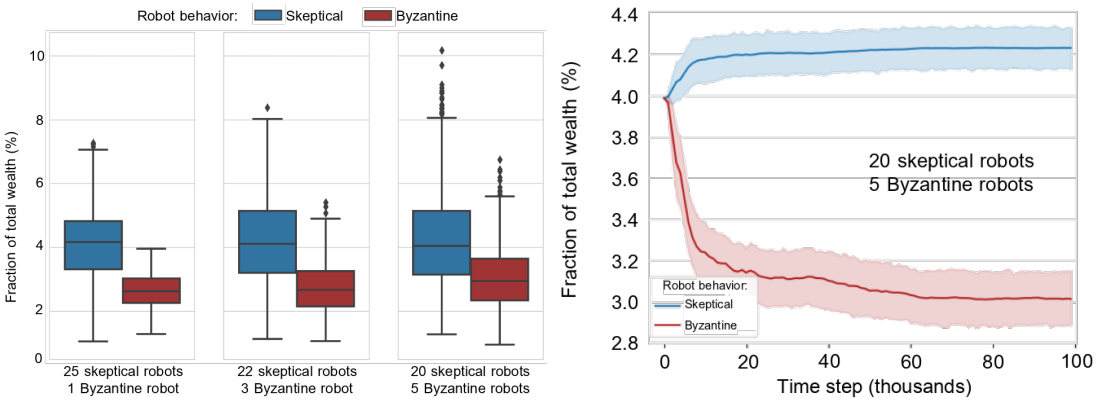


Figure 7.20: Wealth distribution with outlier penalization. The boxplots show the fraction of wealth controlled by each robot after 15 000 simulation steps (128 trials). In swarms of 25, each robot begins with 4% of the total wealth available. Byzantine robots end up with less than this amount; however, this protection becomes less effective as their numbers increase from 1 to 5. The distributions for skeptical robots indicate that some individuals may accumulate larger wealth relative to others. The bottom plot shows that over time, the 5 Byzantine robots stabilize their wealth fraction. Image adapted from Van Calck et al. (2023[†]).

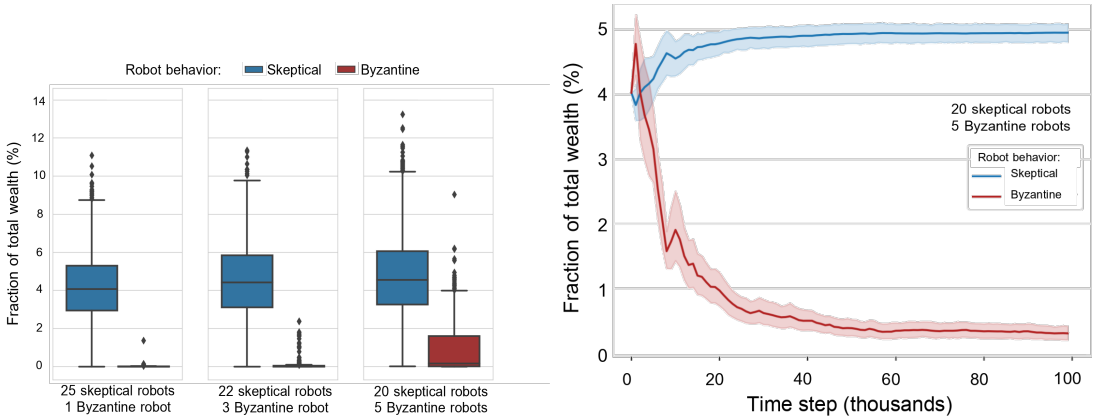


Figure 7.21: Wealth distribution with staking. The boxplots (top) show the fraction of wealth controlled by each robot after 15 000 simulation steps (128 trials). In a swarm of 25, each robot begins with 4% of the total wealth available. Byzantine robots end up with significantly less wealth; however, as their numbers increase, exceptions begin to occur. Over long execution periods (bottom plot), all Byzantine robots converge towards a non-zero wealth fraction, since they are able to maintain an income from foraging themselves that covers their losses. Image adapted from Van Calck et al. (2023[†]).

(C) Outlier penalization with staking

We further enhance the system with a *staking mechanism*: when robots sell information, they must deposit a fixed amount of σ crypto tokens (we used $\sigma = 0.04$). The stake is locked by the smart contract until the buyer finalizes its round trip, at which point the contract transfers rewards plus deposits using the same w_i weights as before, except that the total pool of tokens to distribute is $\frac{R}{2} + |T|\sigma$ (the reward plus the total deposits).

As a consequence, sellers whose reports are classified as outliers not only receive a smaller share, but may also lose net tokens when the stake exceeds their redistributed amount. The mechanism additionally protects the system from Sybil attacks by limiting how many sales a robot can make as a function of its wealth.

Figure 7.21 shows that adding staking increases the separation in wealth between honest and Byzantine robots relative to the no-staking case above. Over the duration of the experiments, Byzantine robots accumulate wealth substantially more slowly than honest robots since they lose tokens for contributing incorrect social information, but can still gain tokens by foraging resources, albeit at a slower rate. The resulting wealth gap between honest and Byzantine robots could be leveraged in future work as a trust signal, allowing robots to weigh or filter information based on the seller's accumulated wealth.

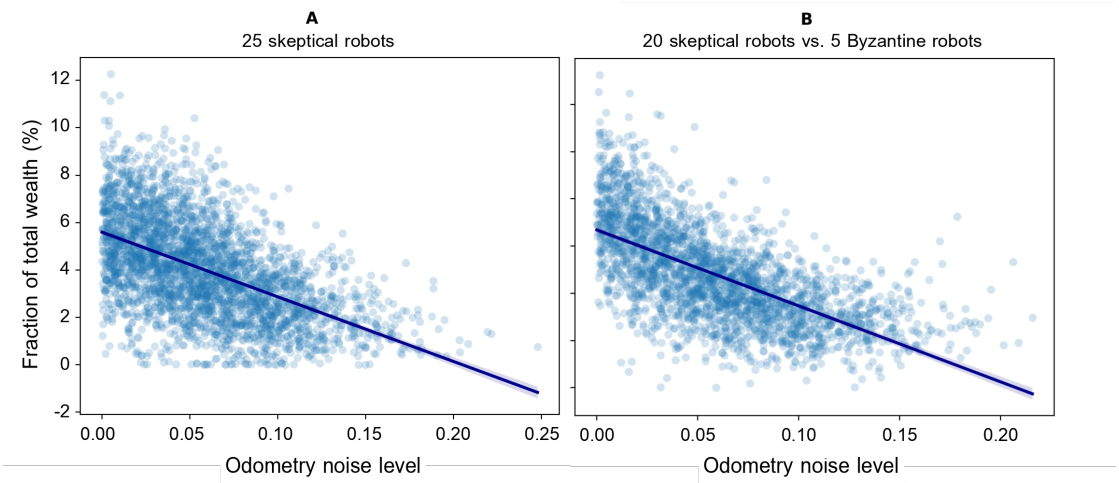


Figure 7.22: Relationship between odometry noise and robots' wealth. We observe results after 15 000 simulation steps (32 trials). The wealth acquired by robots shows Pearson correlation coefficients of -0.54 and -0.64 for the cases with no Byzantine robots and with 5 Byzantine robots, respectively. Image adapted from Van Calck et al. (2023†).

(D) Robot quality versus wealth acquired

Robots' wealth can also indicate their contribution to the collective task. In our system, robots become wealthy by completing round trips or contributing to others' foraging efforts. Figure 7.22 shows a correlation between odometry noise levels and wealth: lower odometry noise is more frequent among the wealthiest robots at the end of the experimental trials. We found a negative Pearson correlation coefficient between robots' odometry noise standard deviation and their wealth (-0.54 for Byzantine-free swarms and -0.64 for swarms with 5 Byzantine robots). As such, wealth can be a valid metric that captures robots' navigation capabilities.

7.5.5 Discussion

Although market-based approaches have been explored in the robotics literature, they are less common in swarm robotics since digital economic exchanges have traditionally required a central entity to settle transactions, host auctions, or enforce contracts. Here, we enable a decentralized information market in a robot swarm by using a blockchain. We have shown how economic interactions can lead to desired global behaviors and how robots' performance, quality, and trustworthiness can be measured by their accumulated wealth. Through multi-agent simulations, we provide a proof of concept for this design approach to regulate robots' social navigation during the execution of central place foraging.

This is an initial step towards an *economics-based design paradigm* for swarms, which may help overcome challenges by leveraging economic theory to study the convergence of collectives towards stable and desired behaviors. For example, one could use Nash equilibria to characterize stable collective behaviors, or apply mechanism design—for instance, mechanisms based on Vickrey–Clarke–Groves (VCG) theory (Groves 1973)—to align robots' interests towards truthful sharing of private information and achieve efficient allocations of shared resources and task assignments. Recently, Gautier et al. (2022) presented a path-finding and negotiation algorithm based on VCG auctions to let robots efficiently and robustly plan their trajectories. Our work can also be extended by increasing the realism of the experiments, for instance, using physical robots, physics-based simulators, or by studying more complex scenarios than two-location foraging (e.g., robots could navigate unknown environments by guiding each other towards waypoints).

Work on *Byzantine fault tolerance* in robotics has not yet addressed open robot swarms: permissionless systems composed of robots from different owners. Open swarms are heterogeneous, comprising robots with different abilities and specifications that are built and programmed by different producers. However, open systems negate many traditional security solutions (Bijani and Robertson 2014), and coordinating such systems, especially at large scales, remains a topic of future research. We identify blockchain technology as key to enabling advances in open swarm robotics research, as it assists in providing solutions to several problems in open multi-agent systems, including Sybil attacks (Jin et al. 2022).

Unlike the work presented in previous chapters, where the blockchain enabled robots to securely update a shared database, here we studied the case where robots are not simply moving sensors; rather, they act as economic agents that seek to coordinate with each other in a physical environment, using the blockchain as a record of one-to-one economic exchanges: the social information exchanged by robots is only relevant to the buyer and the seller, and not to any other robots. While the exchange is public, privacy is maintained since the information is expressed in the buyer's relative coordinate system and is not useful to other robots.

While simple buy-and-sell mechanisms are novel, they do not provide security among mutually untrusting robots, even when recorded on the blockchain. Dedicated mechanisms to mitigate malicious or greedy participants are required. We showed how smart contracts can be employed to enhance Byzantine fault tolerance by enforcing rules for peer-to-peer exchanges, penalizing Byzantine robots, securing the system, and increasing the swarm's efficiency.

Overall, our results show the potential of economics-inspired swarm robotics. Traditional economics, typically steered by central banking, marketplaces, and auctions, has not been considered a suitable inspiration source for the design of robot swarms. Using a blockchain, we introduced the opportunity to enforce system-

wide rules in a fully decentralized and trustless manner. This approach can enhance swarm robotics, extending existing approaches—typically inspired by insect and animal behaviors—to accommodate scenarios where robots are not cooperative by design or might behave strategically. Taking inspiration from economics to design decentralized open swarms is a research direction that we believe is at the beginning of a long and fruitful journey.

7.6 Chapter summary

This chapter demonstrated how the mechanisms developed throughout the thesis translate into concrete applications in robot swarms. Across four case studies, we used a blockchain as a shared and tamper-resistant substrate to (i) coordinate collective decisions without central control, (ii) maintain system robustness and data consistency under intermittent connectivity, and (iii) introduce incentive mechanisms that shape robots' behaviors and long-term resilience.

In Section 7.2, we showed that smart contracts can implement real-time decentralized supervision for foraging by aggregating information about discovered resources and allocating recruits. This application illustrates how on-chain logic can reduce physical interference by shaping swarm-level task allocation while remaining compatible with decentralized operation. In Section 7.3, we used the blockchain as a conflict-free shared data structure for federated learning and implemented aggregation and defenses directly in a smart contract. The results show how our protection mechanisms improve the robustness of swarm learning even when many robots attempt model poisoning; however, the system remains vulnerable to smart Byzantine robots that use available information about peer models and past updates to unfairly gain reputation tokens. In Section 7.4, we improved the robustness of collaborative mapping by validating inter-robot loop closures on-chain before they are used for pose graph optimization. The triangle-based validation and security-level parameter illustrate a trade-off between robustness (including resistance to collusion) and latency. However, it highlights the potential of robot swarms to enhance robustness through peer validation and cooperation. Finally, in Section 7.5, we explored market-based coordination for social navigation, where smart contracts settle peer-to-peer information trades and enforce incentives that promote honesty. This application connects robustness to economic design: token rewards and penalties can align individual profit-seeking behavior with swarm-level reliability, but require careful mechanism design to limit exploitability.

Taken together, these applications support the thesis perspective (Chapter 3): blockchains and smart contracts can serve as general-purpose coordination and security primitives for open swarms. At the same time, they introduce non-trivial costs—notably latency, bandwidth, and storage overheads—which must be balanced

against the desired level of robustness and the capabilities of the target swarm platform on a per-application basis.

Chapter 8

Conclusions and Future Work

This thesis has shown that blockchain technology can provide a principled foundation for secure and fault-tolerant self-organization in robot swarms. It does so through *decentralized oracle mechanisms* that enable robots to validate and agree on non-deterministic updates to a shared state.

The proof-of-concept experiments involved numerous physical and simulated robots (Chapter 5), demonstrating the feasibility of the approach on resource-constrained robots (Figures 5.9 and 5.10) and swarm-wide blockchain synchronization (Figure 5.8).

Swarm Oracle (Chapter 6) implemented Byzantine fault-tolerant consensus in a robot swarm, ensuring trustless coordination that withstands the presence of Byzantine robots that behave arbitrarily or non-cooperatively—provided that the number of Byzantine robots remains within established limits (Figures 6.3 and 6.7). Such guarantees are usually studied in theoretical settings, making their realization in a decentralized swarm of physical robots a significant contribution.

Through four application-focused case studies (Chapter 7), the thesis demonstrated that smart contracts and tailored oracle mechanisms can provide a substrate for coordinating collective behavior, synchronizing and securing shared data, and implementing crypto-token economies. These applications show how blockchain-based methods can bridge the gap between research and real-world applications without compromising the decentralization and fault tolerance of swarm robotics.

Throughout the thesis, we demonstrate that the integration of blockchain technologies with swarm robotics does not necessarily degrade core swarm properties; rather, the results presented in this thesis show how it can enhance them. In Section 8.1, the discussion turns to how the contributions of this thesis affect the three properties central to swarm robotics identified in Chapter 2. Finally, Section 8.2 provides guidelines for future work.

8.1 System properties

Scalability, flexibility, and fault tolerance are widely regarded as the defining properties of swarm robotic systems (Dorigo et al. 2014). The following discussion reflects on how the blockchain-based methods developed in this thesis affect each of these properties.

8.1.1 Scalability

The scalability study in Chapter 5 showed that storage costs grow linearly with time and swarm size, while communication, computation, and runtime memory usage remain stable and evenly distributed across the swarm, without significant bottlenecks or spikes (Section 5.3.6 and Figures 5.9 and 5.10). These results support the claim that hardware resources are not the immediate bottleneck to scalability. Instead, limitations of the Clique proof-of-authority consensus protocol were identified when it was deployed in swarm settings. Figure 5.8 shows that block propagation time increases to approximately 40–60 s in large swarms, while the block period remains fixed at 15 s, causing block production efficiency to fall below 20% as the protocol produces competing blocks that are ultimately discarded. Despite this inefficiency, the blockchain exploits metadata and lightweight synchronization exchanges among robots, reducing the impact and cost of network partitioning. More importantly, the swarm eventually converges to a single blockchain history, which is sufficient for the scenarios considered in this thesis. Efficiency could be improved by using messenger robots to facilitate block dissemination, by selecting parameters more carefully, or by implementing consensus protocols better suited to swarm networks. These directions are discussed further in Section 8.2.

Scalable fault tolerance is a key feature of Swarm Oracle. In our experiments, correct agreements were maintained in the presence of up to four attackers, corresponding to the protocol's one-third threshold (Section 6.3 and Figure 6.7). Because this threshold is defined globally in terms of reputation levels, rather than local neighborhood conditions, the absolute number of tolerated faulty robots increases with swarm size (Section 6.4). A related observation emerges from the collaborative mapping experiments. Section 7.4.7 shows that, in an eight-robot swarm, fewer than half of the proposed loop closures are validated, and only a small fraction reach higher security levels (Figure 7.15). This indicates that small swarms may have limited capacity for repeated cross-validation. To enhance fault tolerance, larger swarms are therefore attractive: with more robots, cross-validation of loop closures would occur more frequently than in small teams. In this way, larger swarms can leverage their "many eyes" to produce systems whose fault tolerance improves with system size.

In Swarm Oracle, the main scalability concern is the requirement that a large proportion of the swarm—more specifically, a supermajority of reputation-token holders—participate in consensus. In general, the trustless design presented in Chapter 6 can be understood as a deliberately conservative choice. It was adopted to maximize fault tolerance in scenarios that are rarely considered in the swarm robotics literature, such as multi-stakeholder systems and adversarial outdoor deployments. Scalability could be improved through committee selection, parallelization of consensus tasks, or reduced quorum sizes. However, these strategies may come at the cost of weaker Byzantine fault tolerance, which may nevertheless be acceptable in non-adversarial swarm deployments.

Chapter 7 shows that application-specific requirements can introduce additional scalability challenges. For instance, many classical task-allocation methods have combinatorial complexity, making them impractical for on-chain execution at swarm scale. In contrast, Section 7.2.5 and Figure 7.3 show that simple blockchain-based rules can improve foraging performance as the swarm grows, although performance remains constrained by the limited availability of environmental resources. Sections 7.3 and 7.4 examine realistic applications that generate large models or depend on the optimization of shared maps, both of which consume substantial on-chain storage and computation. Such applications require dedicated mechanisms to prevent costs from increasing as robots accumulate data over time or as map complexity grows.

8.1.2 Flexibility

In this thesis, swarm-wide action is guided by global information produced through decentralized oracle mechanisms. While this paradigm offers clear potential for secure swarm design and control, Section 7.2.6 and Figure 7.4 show that the performance of this approach can vary across environments, suggesting that fixed global rules provide only limited flexibility. A similar conclusion emerges from the generality results in Chapter 5. Although good performance is observed in randomly generated environments (Section 5.3 and Figure 5.5), Strobel et al. (2020) report that when black floor tiles are concentrated in a single region of the arena, there may be no "one-size-fits-all" solution for achieving consensus. This indicates that global agreements may overlook important local variation and, as a result, reduce flexibility.

A promising direction for future work is to integrate adaptive mechanisms directly into the smart contract, allowing it to learn from the environment and adjust its policies over time. For example, the contract could adapt task-allocation rules based on observed swarm performance or dynamically reallocate resources as robots develop reputations or specialize in particular tasks. At the same time, smart contract policies should preserve a balance between collective coordination

and individual autonomy. Rather than imposing rigid global control, they should shape swarm behavior in ways that improve coordination and efficiency while still allowing local decision-making, exploration, and adaptation by individual robots.

Blockchain-based smart contracts provide a general distributed computation framework for swarm robotics applications. In this thesis, several application-specific contracts were developed, illustrating this generality. Swarm Oracle was likewise designed as a general-purpose protocol that can be adapted to different tasks. Its modular architecture—including interchangeable clustering algorithms, distance functions, and reputation equations (Section 6.5)—allows it to be configured for different observation spaces and application settings, as shown by Zhao et al. (2025†, 2023†), who applied the framework to distinct consensus problems. A natural next step is to develop mechanisms that leverage this generality to enhance flexibility. For example, Swarm Oracles could reconfigure automatically in response to application demands or to the type of data requested on-chain. Smart contracts could also be deployed during online operation to shape swarm behavior, either by humans or by the robots themselves.

8.1.3 Fault tolerance

Robot swarms are naturally resilient to non-malicious faults through redundancy and local adaptation. This thesis tackles the more challenging problem of Byzantine fault tolerance, where robots may behave arbitrarily. The proposed mechanisms address this problem by leveraging the synchronization and deterministic execution of an underlying blockchain to implement Byzantine fault-tolerant (BFT) oracle protocols. This hybrid design provides *safety and liveness guarantees*, although finality remains *probabilistic*: confidence in consensus grows as additional blocks are appended. This is a necessary trade-off for swarm robotics, given its sparse communication patterns and asynchronous interactions.

Probabilistic finality was empirically observed in the scalability study of Chapter 5. Figure 5.8 shows that, in large swarms, block propagation times can exceed the block period, leading to an increased number of blockchain forks. Rather than indicating failure, this behavior is consistent with the underlying consensus model, in which conflicting states may temporarily coexist before converging to a single history.

With respect to agreement on oracle inputs, Chapter 5 first showed how the influence of Byzantine robots can be mitigated. Section 5.3.1 and Figure 5.2 show that recording individual robot contributions on-chain and synchronizing shared-state updates reduces sensitivity to network partitioning relative to typical methods (LeBlanc et al. 2013). However, the proposed mechanism, based on outlier rejection alone and a static quorum size, does not provide formal guarantees, as coordinated adversaries controlling multiple robots may still disrupt the protocol.

To address this limitation, Chapter 6 introduces a reputation-weighted BFT protocol that withstands coordinated attacks while providing explicit safety and liveness guarantees when Byzantine robots control less than one-third of the reputation tokens. Experimental results confirm its *robustness* against safety, liveness, combined, and physical attacks. As shown in Figure 6.7, consensus error remains stable and agreement costs stay within practical bounds. These findings demonstrate that global BFT consensus can be achieved using only peer-to-peer communication in a swarm.

Across the thesis, reputation systems and token economies emerge as key mechanisms for *resilience*. By maintaining a persistent record of robot behavior, the blockchain enables regulation of participation, incentivization of honest contributions, and gradual exclusion of unreliable agents. The proof-of-concept study, Section 5.3.4 and Figure 5.6, shows Byzantine robots experiencing diminishing token inflow over time. The Swarm Oracle results, Figures 6.9 and 6.10, further demonstrate how attackers lose reputation while consensus efficiency improves.

Task-specific evaluations in Chapter 7 show that these mechanisms can support robustness and resilience in several applications.

In federated learning, the proposed mechanisms are shown to be robust to multiple forms of poisoning attacks (Sections 7.3.5 to 7.3.7). However, the results also reveal a limitation: strategic behaviors can exploit the reputation system by predicting the model, thereby gaining tokens without contributing meaningfully to the training process.

In collaborative mapping, robustness is achieved by admitting only loop closures that satisfy geometric consistency constraints. As shown in Section 7.4.6 and Figure 7.16, trajectory error remains close to ground truth even as Byzantine robots are introduced. The system tolerates both benign and malicious faults under the assumption of non-collusion, while robustness to coordinated attacks can be increased through the security level parameter (Section 7.4.4).

In market-based social navigation, Section 7.5.2 and Figure 7.19 show that the baseline system is vulnerable to misleading information. However, reward sharing, outlier penalization, and staking mechanisms progressively shift wealth distribution away from Byzantine robots (Section 7.5.4 and Figures 7.20 and 7.21). In this context, accumulated tokens function as a trust metric grounded in task performance.

Overall, these results demonstrate that blockchain-based oracle mechanism reduce reliance on individual robots and enable robust operation. This expands the applicability of robot swarms beyond controlled environments toward open, multi-stakeholder systems in which cooperative behavior cannot be assumed.

8.2 Future work

The contributions of this thesis open several directions for future research. The following sections outline five areas that are particularly promising.

8.2.1 Tailoring permissioned consensus

This thesis used Ethereum's proof-of-authority (Clique) consensus protocol, which was designed for general-purpose permissioned blockchain networks rather than for the communication patterns and constraints typical of robot swarms.

Developing consensus protocols tailored to swarm robotics could significantly improve efficiency. Several directions are worth exploring. First, *adaptive block periods* that adjust dynamically based on network connectivity could improve efficiency. Blocks could be produced faster when the swarm network is well connected and disseminates blocks quickly, and slow down when connectivity degrades. This would promote faster decision-making, for example, when robots agglomerate at the nest, and slower decision-making while robots are dispersed in the environment performing their assigned tasks.

Second, *mempool management* strategies could quantify transaction importance (e.g., based on age or peer reviews) to prioritize relaying high-value information. Recent protocols such as Narwhal&Tusk (Danezis et al. 2022) combine a transaction dissemination layer based on directed acyclic graphs (DAGs) with a separate consensus layer that provides finality via a permissioned mechanism. Similar approaches could be adapted to swarm applications.

Third, the need for full Byzantine *fault tolerance* should be evaluated on a per-application basis: in cooperative swarms where robots are manufactured and programmed by a single stakeholder, weaker models (e.g., crash faults) may suffice at lower cost, whereas open swarms require stronger guarantees.

Fourth, *succinct blockchains*—in which cryptographic proofs (e.g., zero-knowledge proofs) allow robots to verify blockchain state without downloading and replaying the entire history—could reduce storage and synchronization costs. Such techniques are actively studied in public blockchains for IoT and mobile settings and could benefit resource-constrained robotic platforms (Bonneau et al. 2020).

Finally, *alternative ledger structures* such as DAG-based ledgers (Danezis et al. 2022, Tran et al. 2019) offer partition tolerance and higher throughput, which may better match swarm communication patterns. Their applicability to swarm robotics, however, remains to be demonstrated in practice.

8.2.2 Heterogeneous swarms

The experiments in this thesis employed homogeneous swarms of identical robots. Real-world deployments, however, will likely involve heterogeneous swarms comprising robots with different sensing modalities, computational capabilities, and locomotion types. Heterogeneity introduces both opportunities and challenges for blockchain-based coordination.

On the opportunity side, more capable robots could serve as full blockchain nodes—producing and validating blocks—while simpler robots operate as light clients that verify inclusion of their transactions without maintaining the full chain. This tiered architecture could extend blockchain-based coordination to swarms that include minimalist platforms alongside more powerful ones, mirroring the full-node/light-client distinction in public blockchain networks.

On the challenge side, heterogeneity complicates the design of smart contracts and reputation systems. Robots with better sensors may consistently accrue higher reputation, potentially marginalizing less capable but still functional peers. Designing mechanisms that account for and leverage heterogeneous capabilities—for example, assigning different roles or weighting contributions based on demonstrated competence rather than uniform expectations—is an important direction for future work. Initial steps in this direction have been taken by Zhao et al. (2025†), who studied how heterogeneous capabilities can be leveraged within the Swarm Oracle framework to improve collective performance. In that work, faster robots are incentivized to use mobility to discover resources, while slower robots with more accurate position estimates act as validators to support accurate positioning.

8.2.3 Open swarms and Sybil protection

The concept of *open swarms*—robot collectives composed of robots from different stakeholders that join and leave dynamically—was discussed on multiple occasions in this thesis. Open swarms, however, introduce challenges that remain to be addressed.

A critical open problem is *Sybil protection*: preventing a malicious actor from creating multiple fake identities to gain disproportionate influence. In the permissioned deployments studied in this thesis, robot identities are established by system administrators, or are assumed to be supported by an external agent (e.g., a trusted stakeholder or the robot's owner) prior to deployment. In open swarms, new mechanisms are needed to govern membership without centralized control. Other challenges include incentive alignment, fair resource allocation, and resilience to free riders that join to benefit from collective efforts without contributing.

Possible approaches include proof-of-physical-work protocols, where the right to participate is earned by completing verifiable tasks (Hoffmann 2022, Strobel et al.

2018); stake-based admission, where stakeholders vouch for their robots through locked crypto tokens; and decentralized identity verification via peer attestation, where existing swarm members collectively verify the physical presence and capabilities of newcomers. Decentralized Autonomous Organizations (DAOs), discussed in Chapter 3, offer a governance framework for encoding admission, departure, and exclusion rules in smart contracts, enabling transparent and autonomous self-governance of open swarms. Tang et al. (2025) recently proposed a resilient consensus mechanism for open systems that supports agents joining or leaving the network. Similar alternatives that avoid computational proof-of-work, which is unsuitable for resource-constrained devices, should be investigated.

8.2.4 Privacy and transparency

Blockchains are transparent by design: all transactions and contract states are visible to every participant to enable trustless verification and bottom-up fault tolerance. However, as discussed in the proof-of-concept vulnerability analysis (Section 5.4), in the Swarm Oracle vulnerability analysis (Section 6.4), and in the federated learning smart-attacker experiments (Section 7.3.7 and Figures 7.10 and 7.11), this transparency can also be exploited. In the proof-of-concept study, Byzantine robots could eavesdrop on peers' estimates to craft submissions that evade outlier detection. In Swarm Oracle, copycats could replicate the observations of reputable robots to avoid losing reputation for inactivity. In federated learning, smart Byzantine robots exploited public model information to gain reputation without performing useful work.

Addressing this tension between transparency and privacy is an important direction for future research. *Commit-reveal schemes*, in which robots first submit cryptographic commitments and reveal their data only after a collection phase, can prevent eavesdropping but introduce additional communication rounds. *Zero-knowledge proofs* could allow robots to prove properties of their contributions (e.g., that a model update improves performance) without revealing the underlying data, though computational costs must be carefully evaluated for resource-constrained platforms.

Another approach is *locality-sensitive hashing (LSH)*, which enables detection of similar data without fully disclosing it (Datar et al. 2004, Jafari et al. 2021). Instead of publishing raw data (model updates, sensor readings, trajectories, etc.), participants could publish similarity-preserving hashes that support coordination and verification while reducing information leakage. These hashes enable approximate duplicate detection, clustering, and outlier identification while preserving privacy.

Overall, balancing transparency (needed for trustless verification and accountability) with privacy (needed to prevent exploitation) is a design challenge that will

become increasingly important as blockchain-based swarms move toward real-world deployments, especially in multi-stakeholder settings.

8.2.5 Human interactions

This thesis focused on fully autonomous robot swarms that operate without human intervention during deployment. Real-world applications, however, will require meaningful interaction between humans and blockchain-enabled swarms. This interaction can take several forms.

First, *human-in-the-loop oversight*: the blockchain's immutable, replicated log provides a natural interface for external monitoring and auditing. Retrieving the blockchain from a single robot enables reconstruction of the complete history of operations, supporting compliance verification and liability assignment. Developing intuitive interfaces for non-expert users to query, visualize, and interpret blockchain-recorded swarm behavior is an important practical direction.

Second, *human participation in governance*: in open swarms, stakeholders—individuals, companies, or governmental bodies—may need to participate in governance decisions such as mission specification, resource allocation, or conflict resolution. Smart contract-based governance mechanisms (DAOs) could be extended to include human participants alongside robots, enabling hybrid decision-making that combines human judgment with robotic autonomy.

Third, *human-robot economic interactions*: as discussed in Chapter 3, blockchains enable economic interactions in which humans hire robot swarms for tasks and compensate them via crypto payments, supporting robots-as-a-service business models. Conversely, robots could autonomously pay for maintenance, energy, or cloud services. Designing these economic interfaces to be intuitive, fair, and accountable is essential for adoption of blockchain-enabled swarm robotics in real-world applications.

Bibliography

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng (2016). "TensorFlow: A system for large-scale machine learning". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, pp. 265–283.

Aditya, S., R. Singh, P. K. Singh, and A. Kalla (2021). "A Survey on Blockchain in Robotics: Issues, Opportunities, Challenges and Future Directions". In: *Journal of Network and Computer Applications* 196, p. 103245.

Agmon, N. and D. Peleg (2006). "Fault-Tolerant Gathering Algorithms for Autonomous Mobile Robots". In: *SIAM Journal on Computing* 36.1, pp. 56–82.

Albert, R., H. Jeong, and A.-L. Barabási (2000). "Error and attack tolerance of complex networks". In: *Nature* 406.6794, pp. 378–382.

Alsamhi, S. H. and B. Lee (2021). "Blockchain-Empowered Multi-Robot Collaboration to Fight COVID-19 and Future Pandemics". In: *IEEE Access* 9, pp. 44173–44197.

Alsamhi, S. H., A. V. Shvetsov, S. V. Shvetsova, A. Hawbani, M. Guizani, M. A. Alhartomi, and O. Ma (2023). "Blockchain-Empowered Security and Energy Efficiency of Drone Swarm Consensus for Environment Exploration". In: *IEEE Transactions on Green Communications and Networking* 7.1, pp. 328–338.

Amsters, R. and P. Slaets (2020). "Turtlebot 3 as a robotics education platform". In: *Robotics in Education*. Vol. 1023. Advances in Intelligent Systems and Computing. Cham, Switzerland: Springer, pp. 170–181.

Anderson, T. and J. Knight (1983). "A Framework for Software Fault Tolerance in Real-Time Systems". In: *IEEE Transactions on Software Engineering* SE-9.3, pp. 355–364.

Androulaki, E., A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al. (2018). "Hyperledger fabric: a distributed operating system for permissioned blockchains". In: *Proceedings of the 13th EuroSys conference*. ACM Press, pp. 1–15.

Antonic, N., R. Zakir, M. Dorigo, and A. Reina (2024). "Collective Robustness of Heterogeneous Decision-Makers Against Stubborn Individual". In: Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, pp. 68–77.

Antonopoulos, A. M., O. Osuntokun, and R. Pickhardt (2021). *Mastering the lightning network*. Sebastopol, CA, USA: O'Reilly Media. ISBN: 9781492054818.

Antonopoulos, A. M. and G. Wood (2018). *Mastering Ethereum: Implementing Smart Contracts*. Sebastopol, CA, USA: O'reilly Media. ISBN: 9781491971932.

Aswale, A., A. López, A. Ammartayakun, and C. Pinciroli (2022). "Hacking the Colony: On the Disruptive Effect of Misleading Pheromone and How to Defend against It". In: *Proceedings of the 21st International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2022)*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 27–34.

Auki (2025). *The Auki whitepaper*. Version v0.8.13. URL: `https://auki.gitbook.io/whitepaper` (visited on 11/11/2025).

Aulinas, J., Y. Petillot, J. Salvi, and X. Lladó (2008). "The SLAM problem: a survey". In: *Artificial Intelligence Research and Development* 184, pp. 363–371.

Ayache, N. and O. Faugeras (1988). "Building, registrating and fusing noisy visual maps". In: *The International Journal of Robotics Research* 7.6, pp. 45–65.

Back, A. (2002). *Hashcash - A Denial of Service Counter-Measure*. URL: `http://www.hashcash.org/papers/hashcash.pdf` (visited on 11/11/2025).

Ball, M., A. Rosen, M. Sabin, and P. N. Vasudevan (2017). *Proofs of useful work*. Cryptology ePrint Archive: `2017/203`. URL: `https://eprint.iacr.org/2017/203`.

Band Protocol (2024). *Band Protocol Whitepaper*. URL: `https://bandprotocol.github.io/whitepaper/doc.pdf` (visited on 11/11/2025).

Barczentewicz, M., A. Sarch, and N. Vasan (2023). "Battle of the Crypto Bots: Automated Transaction Copying in Decentralized Finance". In: *SSRN Electronic Journal* 26.3, pp. 672–730.

Bareilles, G., W. Bouaziz, J. Fageot, and E.-M. El-Mhamdi (2026). "Byzantine Machine Learning: MultiKrum and an optimal notion of robustness". In: arXiv preprint: `2602.03899` (cs.RO). URL: `https://arxiv.org/abs/2602.03899`.

Beckers, R., O. E. Holland, and J.-L. Deneubourg (2000). "From local actions to global tasks: Stigmergy and collective robotics". In: *Prerational Intelligence*. Vol. 26. Studies in Cognitive Systems. Springer, pp. 1008–1022.

Beniiche, A. (2020). *A Study of Blockchain Oracles*. arXiv: `2004.07140 [cs.CR]`. URL: `https://arxiv.org/abs/2004.07140`.

Benligiray, B., S. Milić, and H. Vänttinen (2020). *Decentralized APIs for Web 3.0*. Version v1.0.3. URL: `https://old-docs.api3.org/api3-whitepaper-v1.0.3.pdf` (visited on 11/11/2025).

Bergemann, D. and S. Morris (2005). "Robust Mechanism Design". In: *Econometrica* 73.6, pp. 1771–1813.

Biesmeijer, J. C. and H. de Vries (2001). "Exploration and exploitation of food sources by social insect colonies: A revision of the scout-recruit concept". In: *Behavioral Ecology and Sociobiology* 49.2, pp. 89–99.

Bijani, S. and D. Robertson (2014). "A review of attacks and security approaches in open multi-agent systems". In: *Artificial Intelligence Review* 42, pp. 607–636.

Birk, A. and S. Carpin (2006). "Merging occupancy grid maps from multiple robots". In: *IEEE Proceedings, Special Issue on Multi-Robot Systems* 94.7, pp. 1384–1397.

Bjerknes, J. D. and A. F. T. Winfield (2013). "On Fault Tolerance and Scalability of Swarm Robotic Systems". In: *Distributed Autonomous Robotic Systems*. Springer Tracts in Advanced Robotics. Berlin, Germany: Springer, pp. 431–444.

Blanchard, P., E. M. El Mhamdi, R. Guerraoui, and J. Stainer (2017). "Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent". In: *Advances in Neural Information Processing Systems*. Vol. 30. Neural Information Processing Systems Foundation, pp. 118–128.

Bonabeau, E., M. Dorigo, and G. Theraulaz (1999). *Swarm Intelligence: From Natural to Artificial Systems*. New York: Oxford University Press.

Bonneau, J., I. Meckler, V. Rao, and E. Shapiro (2020). *Mina: Decentralized Cryptocurrency at Scale*. URL: `https://docs.minaprotocol.com/assets/technicalWhitepaper.pdf` (visited on 11/11/2025).

Bouzid, Z., M. G. Potop-Butucaru, and S. Tixeuil (2009a). "Byzantine-Resilient Convergence in Oblivious Robot Networks". In: *Distributed Computing and Networking*. Springer, pp. 275–280.

Bouzid, Z., M. G. Potop-Butucaru, and S. Tixeuil (2009b). "Optimal Byzantine Resilient Convergence in Asynchronous Robots Networks". In: *Stabilization, Safety, and Security of Distributed Systems*. Springer, pp. 165–179.

Bouzid, Z., M. G. Potop-Butucaru, and S. Tixeuil (2010). "Optimal Byzantine-Resilient Convergence in Uni-Dimensional Robot Networks". In: *Theoretical Computer Science* 411.34, pp. 3154–3168.

Brambilla, M., A. Brutschy, M. Dorigo, and M. Birattari (2015). "Property-driven design for robot swarms: A design method based on prescriptive modeling and model checking". In: *ACM Transactions on Autonomous and Adaptive Systems* 9.4, 17:1–17:28.

Brambilla, M., E. Ferrante, M. Birattari, and M. Dorigo (2013). "Swarm Robotics: A Review from the Swarm Engineering Perspective". In: *Swarm Intelligence* 7.1, pp. 1–41.

Breidenbach, L., C. Cachin, B. Chan, A. Coventry, S. Ellis, A. Juels, F. Koushanfar, A. Miller, B. Magauran, D. Moroz, S. Nazarov, A. Topliceanu, F. Tramèr, and F. Zhang (2021). *Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle*

Networks. Version v1.0. URL: https://research.chain.link/whitepaper-v2.pdf (visited on 11/11/2025).

Buterin, V. (2014). *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.* URL: https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf (visited on 11/11/2025).

Buterin, V. (2024). *Possible futures of the Ethereum protocol, part 2: The Surge.* URL: https://vitalik.eth.limo/general/2024/10/17/futures2.html (visited on 11/11/2025).

Buterin, V., D. Feist, D. Loerakker, G. Kadianakis, M. Garnett, M. Taiwo, and A. Dietrichs (2022). *EIP-4844: Shard Blob Transactions*. URL: https://eips.ethereum.org/EIPS/eip-4844 (visited on 11/11/2025).

Cadena, C., L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard (2016). "Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32.6, pp. 1309–1332.

Caldarelli, G. (2020). "Understanding the blockchain oracle problem: A call for action". In: *Information* 11.11, p. 509.

Camazine, S., J.-L. Deneubourg, G. Theraula, J. Sneyd, and N. R. Franks (2020). *Self-Organization in Biological Systems*. Princeton Studies in Complexity. NJ, USA: Princeton University Press, p. 562. ISBN: 9780691212920.

Campo, A., Á. Gutiérrez, S. Nouyan, C. Pinciroli, V. Longchamp, S. Garnier, and M. Dorigo (2010). "Artificial Pheromone for Path Selection by a Foraging Swarm of Robots". In: *Biological Cybernetics* 103.5, pp. 339–352.

Capelli, B., H. Fouad, G. Beltrame, and L. Sabattini (2021). "Decentralized connectivity maintenance with time delays using control barrier functions". In: *IEEE International Conference on Robotics and Automation – ICRA 2021*. IEEE Press, pp. 1586–1592.

Caprari, G., P. Balmer, R. Piguet, and R. Siegwart (1998). "The Autonomous Micro Robot "Alice": a platform for scientific and commercial applications". In: *Proceedings of the 1998 International Symposium on Micromechatronics and Human Science (MHA 1998)*, pp. 231–235.

Cardenas, I. S., J. B. May, and J.-H. Kim (2021). "AutomataDAO: A Blockchain-Based Data Marketplace for Interactive Robot and IoT Data Exchanges Using Ethermint and State Channels". In: *Blockchain Technology for IoT Applications*. Blockchain Technologies. Singapore: Springer, pp. 17–38.

Castelló Ferrer, E., T. Hardjono, A. Pentland, and M. Dorigo (2021a). "Secure and secret cooperation in robot swarms". In: *Science Robotics* 6.56, abf1538.

Castelló Ferrer, E. (2016). *The blockchain: A new framework for robotic swarm systems*. arXiv preprint: 1608.00695 (cs.RO). URL: https://arxiv.org/abs/1608.00695v4.

Castelló Ferrer, E. (2018). "The blockchain: A new framework for robotic swarm systems". In: *Proceedings of the Future Technologies Conference (FTC 2018)*. Springer, pp. 1037–1058.

Castelló Ferrer, E. (2023). "If blockchain is the solution, robot security is the problem". In: *Frontiers in Blockchain* 6, p. 1181820.

Castelló Ferrer, E., I. Berman, A. Kapitonov, V. Manaenko, M. Chernyaev, and P. Tarasov (2023). "Gaka-chu: A self-employed autonomous robot artist". In: *IEEE International Conference on Robotics and Automation – ICRA 2024*. IEEE Press, pp. 11583–11589.

Castelló Ferrer, E., E. Jiménez, J. L. Lopez-Presa, and J. Martín-Rueda (2021b). "Following Leaders in Byzantine Multirobot Systems by Using Blockchain Technology". In: *IEEE Transactions on Robotics* 38.2, pp. 1101–1117.

Castelló Ferrer, E., O. Rudovic, T. Hardjono, and A. Pentland (2018). "Robochain: A secure data-sharing framework for human-robot interaction". In: *Proceedings of the 10th International Conference on eHealth, Telemedicine, and Social Medicine (eTELEMED 2018)*. IARIA Press, pp. 124–130.

Castro, M. and B. Liskov (1999). "Practical Byzantine Fault Tolerance". In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)*. ACM Press, pp. 173–186.

Castro, M. and B. Liskov (2002). "Practical Byzantine Fault Tolerance and Proactive Recovery". In: *ACM Transactions on Computer Systems* 20.4, pp. 398–461.

Cavorsi, M., L. Sabattini, and S. Gil (2024). "Multirobot Adversarial Resilience Using Control Barrier Functions". In: *IEEE Transactions on Robotics* 40, pp. 797–815.

Chang, Y., K. Ebadi, C. E. Denniston, M. F. Ginting, A. Rosinol, A. Reinke, M. Palieri, J. Shi, A. Chatterjee, B. Morrell, A. Agha-mohammadi, and L. Carlone (2022). "LAMP 2.0: A Robust Multi-Robot SLAM System for Operation in Challenging Large-Scale Underground Environments". In: *IEEE Robotics and Automation Letters*, pp. 9175–9182.

Chaum, D., A. Fiat, and M. Naor (1990). "Untraceable Electronic Cash". In: *Advances in Cryptology – CRYPTO '88*. Springer, pp. 319–327.

Chen, W., X. Wang, S. Gao, G. Shang, C. Zhou, Z. Li, C. Xu, and K. Hu (2023). "Overview of Multi-Robot Collaborative SLAM from the Perspective of Data Fusion". In: *Machines* 11.6, p. 653.

Chopra, S., G. Notarstefano, M. Rice, and M. Egerstedt (2017). "A Distributed Version of the Hungarian Method for Multirobot Assignment". In: *IEEE Transactions on Robotics* 33.4, pp. 932–947.

Christensen, A. L., R. O'Grady, M. Birattari, and M. Dorigo (2008). "Fault Detection in Autonomous Robots Based on Fault Injection and Learning". In: *Autonomous Robots* 24.1, pp. 49–67.

Christensen, A. L., R. O'Grady, and M. Dorigo (2009). "From Fireflies to Fault Tolerant Swarms of Robots". In: *IEEE Transactions on Evolutionary Computation* 13.4, pp. 754–766.

Cieliebak, M. (2004). "Gathering non-oblivious mobile robots". In: *Latin American Symposium on Theoretical Informatics*, pp. 577–588.

Cieslewski, T., S. Choudhary, and D. Scaramuzza (2018). "Data-Efficient Decentralized Visual SLAM". In: *IEEE International Conference on Robotics and Automation – ICRA 2018*. IEEE Press, pp. 2466–2473.

Civit, P., M. A. Dzulfikar, S. Gilbert, V. Gramoli, R. Guerraoui, J. Komatovic, and M. Vidigueira (2022). "Byzantine Consensus Is Θ (n^2): The Dolev-Reischuk Bound Is Tight Even in Partial Synchrony!" In: *International Symposium on Distributed Computing (DISC 2022)*, pp. 14–1.

Clearpath Robotics (2025). *TurtleBot 4 User Manual*. Version v2.01. URL: `https://turtlebot.github.io/turtlebot4-user-manual/` (visited on 11/11/2025).

Colosimo, F. and F. De Rango (2023). "Median-Krum: A Joint Distance-Statistical Based Byzantine-Robust Algorithm in Federated Learning". In: *Proceedings of the 21st ACM International Symposium on Mobility Management and Wireless Access (MobiWac'23)*. ACM Press, pp. 61–68.

Connell, J. H. (1990). *Minimalist Mobile Robotics*. First. Vol. 5. Perspectives in Artificial Intelligence. Boston: Morgan Kaufmann. ISBN: 9780080511719.

Cortés, J., S. Martínez, and F. Bullo (2006). "Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions". In: *IEEE Transactions on Automatic Control* 51.8, pp. 1289–1298.

Couzin, I. D. (2009). "Collective cognition in animal groups". In: *Trends in cognitive sciences* 13.1, pp. 36–43.

Cramariuc, A., L. Bernreiter, F. Tschopp, M. Fehr, V. Reijgwart, J. Nieto, R. Siegwart, and C. Cadena (2023). "maplab 2.0 – A Modular and Multi-Modal Mapping Framework". In: *IEEE Robotics and Automation Letters* 8.2, pp. 520–527.

Croman, K., C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, et al. (2016). "On Scaling Decentralized Blockchains: (A Position Paper)". In: *International conference on financial cryptography and data security*, pp. 106–125.

Crowley, J. L. (1989). "World modeling and position estimation for a mobile robot using ultrasonic ranging". In: *IEEE International Conference on Robotics and Automation – ICRA 1989*. Vol. 2, pp. 674–680.

Danezis, G., L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman (2022). "Narwhal and tusk: a dag-based mempool and efficient bft consensus". In: *Proceedings of the Seventeenth European Conference on Computer Systems*, pp. 34–50.

Danilov, K., R. Rezin, I. Afanasyev, and A. Kolotov (2018). "Towards Blockchain-Based Robonomics: Autonomous Agents Behavior Validation". In: *Proceedings of the 9th IEEE International Conference on Intelligent Systems (IS 2018)*. IEEE Press, pp. 222–227.

Datar, M., N. Immorlica, P. Indyk, and V. S. Mirrokni (2004). "Locality-sensitive hashing scheme based on p-stable distributions". In: *Proceedings of the 20th annual symposium on Computational geometry*. New York: ACM Press, pp. 253–262.

De Nicola, R., L. Di Stefano, and O. Inverso (2020). "Multi-agent systems with virtual stigmergy". In: *Science of Computer Programming* 187, p. 102345.

De Vries, A. (2018). "Bitcoin's growing energy problem". In: *Joule* 2.5, pp. 801–805.

Défago, X., M. Gradinariu, S. Messika, and P. Raipin-Parvédy (2006). "Fault-Tolerant and Self-stabilizing Mobile Robots Gathering: –Feasibility Study–". In: *International Symposium on Distributed Computing (DISC 2006)*, pp. 46–60.

Deneubourg, J. .-., S. Aron, S. Goss, and J. M. Pasteels (1990). "The self-organizing exploratory pattern of the argentine ant". In: *Journal of Insect Behavior* 3.2, pp. 159–168. ISSN: 1572-8889.

Deng, G., Y. Zhou, Y. Xu, T. Zhang, and Y. Liu (2021). "An Investigation of Byzantine Threats in Multi-Robot Systems". In: *24th International Symposium on Research in Attacks, Intrusions and Defenses*. New York: ACM Press, pp. 17–32.

Dennis, R., G. Owenson, and B. Aziz (2016). "A Temporal Blockchain: A Formal Analysis". In: *International Conference on Collaboration Technologies and Systems – CTS 2016*. IEEE Press, pp. 430–437.

Di Stasi, G., S. Avallone, R. Canonico, and G. Ventre (2018). "Routing payments on the lightning network". In: *2018 IEEE international conference on internet of things (IThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData)*, pp. 1161–1170.

Dias, M. B. and A. Stentz (2000). "A free market architecture for distributed control of a multirobot system". In: *Proceedings of the 6th International Conference on Intelligent Autonomous Systems (IAS 2000)*, pp. 115–122.

Dias, M. B., R. Zlot, N. Kalra, and A. Stentz (2006). "Market-based multirobot coordination: A survey and analysis". In: *Proceedings of the IEEE* 94.7, pp. 1257–1270.

Dib, O., K.-L. Brousmiche, A. Durand, E. Thea, and E. B. Hamida (2018). "Consortium blockchains: Overview, applications and challenges". In: *Int. J. Adv. Telecommun* 11.1, pp. 51–64.

Dimarogonas, D. V. and K. H. Johansson (2008). "Decentralized connectivity maintenance in mobile networks with bounded inputs". In: *IEEE International Conference on Robotics and Automation – ICRA 2008*, pp. 1507–1512.

Dolev, D. and R. Reischuk (1985). "Bounds on information exchange for Byzantine agreement". In: *Journal of the ACM (JACM)* 32.1, pp. 191–204.

Dorigo, M., M. Birattari, and M. Brambilla (2014). "Swarm Robotics". In: *Scholarpedia* 9.1, p. 1463.

Dorigo, M., 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 (2013). "Swarmanoid: A novel concept for the study of heterogeneous robotic swarms". In: *IEEE Robotics & Automation Magazine* 20.4, pp. 60–71.

Dorigo, M., A. Pacheco, A. Reina, and V. Strobel (2024). "Blockchain technology for mobile multi-robot systems". In: *Nature Reviews Electrical Engineering* 1, pp. 264–274.

Dorigo, M. and T. Stützle (2010). "Ant Colony Optimization: Overview and Recent Advances". In: *Handbook of Metaheuristics, 2nd Edition*. Vol. 146. International Series in Operations Research & Management Science. New York: Springer. Chap. 8, pp. 227–263.

Dorigo, M., G. Theraulaz, and V. Trianni (2020). "Reflections on the future of swarm robotics". In: *Science Robotics* 5.49, abe4385.

Dorigo, M., G. Theraulaz, and V. Trianni (2021). "Swarm Robotics: Past, Present and Future". In: *Proceedings of the IEEE* 109.7, pp. 1152–1165.

Dorigo, M., V. Trianni, E. Şahin, R. Groß, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, and L. M. Gambardella (2004). "Evolving Self-Organizing Behaviors for a Swarm-Bot". In: *Autonomous Robots* 17.2–3, pp. 223–245.

Douceur, J. R. (2002). "The Sybil Attack". In: *1st International Workshop on Peer-to-Peer systems*. Springer, pp. 251–260.

Ducatelle, F., G. Di Caro, A. Förster, M. Bonani, M. Dorigo, S. Magnenat, F. Mondada, R. O'Grady, C. Pinciroli, P. Rétornaz, V. Trianni, and L. M. Gambardella (2014). "Cooperative Navigation in Robotic Swarms". In: *Swarm Intelligence* 8.1, pp. 1–33.

Ducatelle, F., G. A. Di Caro, A. Förster, and L. Gambardella (2010). "Mobile Stigmergic Markers for Navigation in a Heterogeneous Robotic Swarm". In: *Swarm Intelligence*. Springer Berlin Heidelberg, pp. 456–463.

Ducatelle, F., G. A. Di Caro, C. Pinciroli, F. Mondada, and L. Gambardella (2011). "Communication assisted navigation in robotic swarms: Self-organization and cooperation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS 2011*. Piscataway, NJ: IEEE Press, pp. 4981–4988.

Durrant-Whyte, H. and T. Bailey (2006). "Simultaneous localization and mapping: part I". In: *IEEE Robotics & Automation Magazine* 13.2, pp. 99–110.

Dwork, C., N. Lynch, and L. Stockmeyer (1988). "Consensus in the presence of partial synchrony". In: *Journal of the ACM* 35.2, pp. 288–323.

Dwork, C. and M. Naor (1992). "Pricing via processing or combatting junk mail". In: *Proceedings of the Annual International Cryptology Conference – Advances in Cryptology (CRYPTO' 92)*, pp. 139–147.

Edgar, E. (2017). *Reality.eth documentation*. URL: `https://realitio.github.io/docs/html/` (visited on 11/11/2025).

Ekparinya, P., V. Gramoli, and G. Jourjon (2020). "The Attack of the Clones Against Proof-of-Authority". In: *Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS 2020)*. The Internet Society, p. 24082.

El Faqir, Y., J. Arroyo, and S. Hassan (2020). "An Overview of Decentralized Autonomous Organizations on the Blockchain". In: *Proceedings of the 16th International Symposium on Open Collaboration*. ACM Press, pp. 1–8.

Ellis, S., A. Juels, and S. Nazarov (2017). *Chainlink: A Decentralized Oracle Network*. URL: `https://chain.link/whitepaper` (visited on 11/11/2025).

Engel, J. J., T. Schöps, and D. Cremers (2014). "LSD-SLAM: Large-Scale Direct Monocular SLAM". In: *European Conference on Computer Vision*. Vol. 8690. Lecture Notes in Computer Science, pp. 834–849.

Ethereum Foundation (2025). *Ethereum Archive Nodes*. URL: `https://ethereum.org/developers/docs/nodes-and-clients/archive-nodes/` (visited on 11/11/2025).

Fernandez-Cortizas, M., H. Bavle, D. Perez-Saura, J. L. Sanchez-Lopez, P. Campoy, and H. Voos (2024). "Multi S-Graphs: An Efficient Distributed Semantic-Relational Collaborative SLAM". In: *IEEE Robotics and Automation Letters* 9.6, pp. 6004–6011.

Ferrante, E., A. E. Turgut, C. Huepe, A. Stranieri, C. Pinciroli, and M. Dorigo (2012). "Self-Organized Flocking with a Mobile Robot Swarm: a Novel Motion Control Method". In: *Adaptive Behavior* 20.6, pp. 460–477.

Fischer, M. J., N. A. Lynch, and M. S. Paterson (1985). "Impossibility of distributed consensus with one faulty process". In: *Journal of the ACM* 32.2, pp. 374–382.

Flocchini, P. and G. Prencipe (2012). "Distributed Computing by Oblivious Mobile Robots". In: *Synthesis Lectures on Distributed Computing Theory* 3, pp. 1–185.

Flocchini, P., G. Prencipe, N. Santoro, and P. Widmayer (2005). "Gathering of asynchronous robots with limited visibility". In: *Theoretical Computer Science* 337.1, pp. 147–168.

Flocchini, P., G. Prencipe, N. Santoro, and P. Widmayer (2008). "Arbitrary pattern formation by asynchronous, anonymous, oblivious robots". In: *Theoretical Computer Science* 407.1-3, pp. 412–447.

Font Llenas, A., M. S. Talamali, X. Xu, J. A. R. Marshall, and A. Reina (2018). "Quality-Sensitive Foraging by a Robot Swarm Through Virtual Pheromone Trails". In: *Swarm Intelligence – Proceedings of ANTS 2018 – 11th International Conference*. Vol. 11172. Lecture Notes in Computer Science. Cham, Switzerland: Springer, pp. 135–149.

Fox, D., W. Burgard, H. Kruppa, and S. Thrun (2000). "A probabilistic approach to collaborative multi-robot localization". In: *Autonomous robots* 8.3, pp. 325–344.

Fragapane, G., R. De Koster, F. Sgarbossa, and J. O. Strandhagen (2021). "Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda". In: *European journal of operational research* 294.2, pp. 405–426.

Francesca, G., M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari (2014). "AutoMoDe: A novel approach to the automatic design of control software for robot swarms". In: *Swarm Intelligence* 8.2, pp. 89–112.

Fujisawa, R., S. Dobata, K. Sugawara, and F. Matsuno (2014). "Designing pheromone communication in swarm robotics: Group foraging behavior mediated by chemical substance". In: *Swarm Intelligence* 8.3, pp. 227–246.

Fung, C., C. J. M. Yoon, and I. Beschastnikh (2020). "The Limitations of Federated Learning in Sybil Settings". In: *Proceedings of the 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. USENIX Association, pp. 301–316.

Galton, F. (1907). "Vox Populi". In: *Nature* 75, pp. 450–451.

Garattoni, L. and M. Birattari (2016). "Swarm robotics". In: *Wiley Encyclopedia of Electrical and Electronics Engineering*. Hoboken, NJ: John Wiley & Sons, pp. 1–19.

Garattoni, L., A. Ligot, K. Hasselmann, D. Garzón Ramos, G. Francesca, C. Pinciroli, and M. Birattari (2021). *demiurge-project/argos3-epuck*. Version v48. URL: `https://doi.org/10.5281/zenodo.4882715`.

Garnier, S., J. Gautrais, M. Asadpour, C. Jost, and G. Theraulaz (2009). "Self-organized aggregation triggers collective decision making in a group of cockroach-like robots". In: *Adaptive Behavior* 17.2, pp. 109–133.

Gautier, A., A. Stephens, B. Lacerda, N. Hawes, and M. Wooldridge (2022). "Negotiated Path Planning for Non-Cooperative Multi-Robot Systems". In: *Proceedings of the 21st International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2022)*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, pp. 472–480.

Gerkey, B. P. and M. J. Mataric (2002). "Sold!: Auction methods for multirobot coordination". In: *IEEE transactions on robotics and automation* 18.5, pp. 758–768.

Gerkey, B. P. and M. J. Matarić (2004). "A formal analysis and taxonomy of task allocation in multi-robot systems". In: *The International journal of robotics research* 23.9, pp. 939–954.

Gielis, J., A. Shankar, and A. Prorok (2022). "A critical review of communications in multi-robot systems". In: *Current Robotics Reports* 3.4, pp. 213–225.

Gil, S., S. Kumar, M. Mazumder, D. Katabi, and D. Rus (2017). "Guaranteeing spoof-resilient multi-robot networks". In: *Autonomous Robots* 41.6, pp. 1383–1400.

Givigi Jr., S. N. and H. M. Schwartz (2006). "A Game Theoretic Approach to Swarm Robotics". In: *Applied Bionics and Biomechanics* 3.3, p. 183949.

Grassé, P.-P. (1959). "La reconstruction du nid et les coordinations inter-individuelles chez Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs". In: *Insectes Sociaux* 6, pp. 41–81.

Grey, J., I. Godage, and O. Seneviratne (2020). "Swarm contracts: Smart contracts in robotic swarms with varying agent behavior". In: *Proceedings of the 2020 IEEE International Conference on Blockchain (Blockchain 2020)*. IEEE Press, pp. 265–272.

Grey, J., O. Seneviratne, and I. Godage (2021). "Blockchain-Based Mechanism for Robotic Cooperation Through Incentives: Prototype Application in Warehouse Automation". In: *Proceedings of the 2021 IEEE International Conference on Blockchain (Blockchain 2021)*. IEEE Press, pp. 597–604.

Grisetti, G., R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg (2010). "Hierarchical optimization on manifolds for online 2D and 3D mapping". In: *IEEE International Conference on Robotics and Automation – ICRA 2010*. IEEE Press, pp. 273–278.

Groves, T. (1973). "Incentives in teams". In: *Econometrica: Journal of the Econometric Society* 41.4, pp. 617–631.

Grunspan, C. and R. Pérez-Marco (2018). *Ant routing algorithm for the lightning network*. arXiv: 1807.00151 [cs.DC]. URL: https://arxiv.org/abs/1807.00151.

Gupta, H., V. Strobel, A. Pacheco, E. Ferrante, E. Natalizio, and M. Dorigo (2024). "Group-level Behavioral Switch in a Robot Swarm Using Blockchain". In: *Swarm Intelligence – Proceedings of ANTS 2024 – 14th International Conference*. Vol. 14987. Lecture Notes in Computer Science. Springer, pp. 98–111.

Gupta, V., C. Langbort, and R. M. Murray (2006). "On the robustness of distributed algorithms". In: *Proceedings of the 45th IEEE Conference on Decision and Control*. IEEE Press, pp. 3473–3478.

Gutiérrez, A., A. Campo, M. Dorigo, J. Donate, F. Monasterio-Huelin, and L. Magdalena (2009a). "Open E-puck Range & Bearing Miniaturized Board for Local Communication in Swarm Robotics". In: *IEEE International Conference on Robotics and Automation – ICRA 2009*. Piscataway, NJ: IEEE Press, pp. 3111–3116.

Gutiérrez, A., A. Campo, F. Monasterio-Huelin, L. Magdalena, and M. Dorigo (2010). "Collective Decision-making Based on Social Odometry". In: *Neural Computing & Applications* 19.6, pp. 807–823.

Gutiérrez, A., A. Campo, F. C. Santos, F. Monasterio-Huelin, and M. Dorigo (2009b). "Social Odometry: Imitation Based Odometry in Collective Robotics". In: *International Journal of Advanced Robotic Systems* 26.2, pp. 129–136.

Haeringer, G. (2018). *Market design: auctions and matching*. MIT Press.

Haleem, A., A. Allen, A. Thompson, M. Nijdam, and R. Garg (2024). "Helium: A decentralized wireless communication network". In: *RS Open Journal on Innovative Communication Technologies* 4.13.

Hamann, H. (2018). *Swarm Robotics: A Formal Approach*. Second. Cham, Switzerland: Springer.

Hassan, S. and P. De Filippi (2021). "Decentralized Autonomous Organization". In: *Internet Policy Review* 10.2, pp. 1–10.

Hasselmann, K., A. Parravicini, A. Pacheco, and V. Strobel (2021). *Python wrapper for ARGoS3 simulator*. URL: `https://github.com/KenN7/argos-python/` (visited on 11/11/2025).

Hayek, F. A. (1945). "The Use of Knowledge in Society". In: *The American Economic Review* 35.4, pp. 519–530.

Higgins, F., A. Tomlinson, and K. M. Martin (2009). "Survey on Security Challenges for Swarm Robotics". In: *Proceedings of the Fifth International Conference on Autonomic and Autonomous Systems*. IEEE Press, pp. 307–312.

Hivemapper (2025). *Hivemapper documentation*. URL: `https://docs.hivemapper.com/` (visited on 11/11/2025).

Hoff, N., R. Wood, and R. Nagpal (2013). "Distributed Colony-Level Algorithm Switching for Robot Swarm Foraging". In: *Distributed Autonomous Robotic Systems*. Vol. 83. Springer Tracts in Advanced Robotics. Berlin, Germany: Springer, pp. 417–430.

Hoffmann, F. (2022). "Challenges of Proof-of-Useful-Work (PoUW)". In: *Proceedings of the IEEE 1st Global Emerging Technology Blockchain Forum: Blockchain & Beyond (iGETblockchain 2022)*. IEEE Press, pp. 1–5.

Horton, M., D. Chen, Y. Yi, X. Wen, and J. Doebbler (2023). "GEODNET: Global Earth Observation Decentralized Network". In: *NAVIGATION: Journal of the Institute of Navigation* 70.4.

Houston, A. I. and J. M. McNamara (1985). "A general theory of central place foraging for single-prey loaders". In: *Theoretical Population Biology* 28.3, pp. 233–262. ISSN: 0040-5809.

Huang, Y., T. Shan, F. Chen, and B. Englot (2022). "DiSCo-SLAM: Distributed Scan Context-Enabled Multi-Robot LiDAR SLAM With Two-Stage Global-Local Graph Optimization". In: *IEEE Robotics and Automation Letters* 7.2, pp. 1150–1157.

Hunt, E. R. and S. Hauert (2020). "A checklist for safe robot swarms". In: *Nature Machine Intelligence* 2.8, pp. 420–422.

Isermann, R. (2005). *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. First. Berlin, Germany: Springer.

Jadbabaie, A., J. Lin, and A. S. Morse (2003). "Coordination of groups of mobile autonomous agents using nearest neighbor rules". In: *IEEE Transactions on Automatic Control* 48.6, pp. 988–1001.

Jafari, O., P. Maurya, P. Nagarkar, K. M. Islam, and C. Crushev (2021). "A survey on locality sensitive hashing algorithms and their applications". In: arXiv preprint: `2102.08942` (cs.DB). URL: `https://arxiv.org/abs/2102.08942`.

Jin, D., N. Kannengießer, B. Sturm, and A. Sunyaev (2022). "Tackling Challenges of Robustness Measures for Agent Collaboration in Open Multi-Agent Systems". In: *Proceedings of the 55th Hawaii International Conference on System Sciences*. Vol. 7, pp. 7585–7594.

Jones, S. and S. Hauert (2025). "Distributed spatial awareness for robot swarms". In: *Autonomous Robots* 49.4, p. 41.

Kapitonov, A., S. Lonshakov, V. Bulatov, B. K. Montazam, and J. White (2021). "Robot-as-a-Service: From Cloud to Peering Technologies". In: *Frontiers in Robotics and AI* 8, p. 560829. ISSN: 2296-9144.

Kapitonov, A., S. Lonshakov, A. Krupenkin, and I. Berman (2017). "Blockchain-based protocol of autonomous business activity for multi-agent systems consisting of UAVs". In: *2017 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*. IEEE Press, pp. 84–89.

Kegeleirs, M., D. Garzón Ramos, and M. Birattari (2019). "Random Walk Exploration for Swarm Mapping". In: *Towards Autonomous Robotic Systems: 20th Annual Conference – TAROS 2019*. Vol. 11650. Lecture Notes in Computer Science. Berlin, Germany: Springer, pp. 211–222.

Kegeleirs, M., G. Grisetti, and M. Birattari (2021). "Swarm SLAM: Challenges and Perspectives". In: *Frontiers in Robotics and AI* 8, p. 618268.

Keramat, F., J. Peña Queralta, and T. Westerlund (2023). "Partition-tolerant and Byzantine-tolerant decision making for distributed robotic systems with IOTA and ROS2". In: *IEEE Internet of Things Journal* 10.14, pp. 12985–12998.

Khadidos, A., R. M. Crowder, and P. H. Chappell (2015). "Exogenous fault detection and recovery for swarm robotics". In: *IFAC-PapersOnLine* 48.3, pp. 2405–2410.

Khaliq, A. A., M. Di Rocco, and A. Saffiotti (2014). "Stigmergic algorithms for multiple minimalistic robots on an RFID floor". In: *Swarm intelligence* 8.3, pp. 199–225.

Khaluf, Y. and H. Hamann (2019). "Modulating Interaction Times in an Artificial Society of Robots". In: *ALIFE 2019: The 2019 Conference on Artificial Life*. The International Society for Artificial Life, pp. 372–379.

Kim, B., M. Kaess, L. Fletcher, J. Leonard, A. Bachrach, N. Roy, and S. Teller (2010). "Multiple relative pose graphs for robust cooperative mapping". In: *IEEE International Conference on Robotics and Automation – ICRA 2010*, pp. 3185–3192.

King, S. (2013). *Primecoin: Cryptocurrency with Prime Number Proof-of-Work*. URL: `https://primecoin.io/primecoin-paper.pdf` (visited on 11/11/2025).

King, S. and S. Nadal (2012). *Ppcoin: Peer-to-peer crypto-currency with proof-of-stake*. URL: `https://www.peercoin.net/read/papers/peercoin-paper.pdf` (visited on 11/11/2025).

Koenig, N. and A. Howard (2004). "Design and use paradigms for Gazebo, an open-source multi-robot simulator". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS 2004*. Vol. 3, pp. 2149–2154.

Kornienko, S., O. Kornienko, and P. Levi (2005). "Minimalistic approach towards communication and perception in microrobotic swarms". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS 2005*. IEEE Press, pp. 2228–2234.

Kuckling, J., R. Luckey, V. Avrutin, A. Vardy, A. Reina, and H. Hamann (2024). "Do we run large-scale multi-robot systems on the edge? More evidence for two-phase performance in system size scaling". In: *IEEE International Conference on Robotics and Automation – ICRA 2024*. IEEE Press, pp. 4562–4568.

Kümmerle, R., B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner (2009). "On measuring the accuracy of SLAM algorithms". In: *Autonomous Robots* 27.4, pp. 387–407.

Kwon, J. (2014). *Tendermint: Consensus without mining*. Version v.0.6. URL: `https://tendermint.com/static/docs/tendermint.pdf` (visited on 11/11/2025).

Lagoudakis, M., M. Berhault, S. Koenig, P. Keskinocak, and A. Kleywegt (2004). "Simple auctions with performance guarantees for multi-robot task allocation".

In: *IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS 2004*. Vol. 1. IEEE Press, pp. 698–705.

Lajoie, P.-Y. and G. Beltrame (2024). "Swarm-SLAM: Sparse Decentralized Collaborative Simultaneous Localization and Mapping Framework for Multi-Robot Systems". In: *IEEE Robotics and Automation Letters* 9.1, pp. 475–482.

Lajoie, P.-Y., S. Hu, G. Beltrame, and L. Carlone (2019). "Modeling Perceptual Aliasing in SLAM via Discrete–Continuous Graphical Models". In: *IEEE Robotics and Automation Letters* 4.2, pp. 1232–1239.

Lajoie, P.-Y., B. Ramtoula, Y. Chang, L. Carlone, and G. Beltrame (2020). "DOOR-SLAM: Distributed, Online, and Outlier Resilient SLAM for Robotic Teams". In: *IEEE Robotics and Automation Letters* 5.2, pp. 1656–1663.

Lajoie, P.-Y., B. Ramtoula, F. Wu, and G. Beltrame (2022). "Towards Collaborative Simultaneous Localization and Mapping: A Survey of the Current Research Landscape". In: *Field Robotics* 2, pp. 971–1000.

Lamport, L. (1977). "Proving the Correctness of Multiprocess Programs". In: *IEEE Transactions on Software Engineering* SE-3.2, pp. 125–143.

Lamport, L. (2011). "Byzantizing Paxos by refinement". In: *International symposium on distributed computing (DISC 2011)*, pp. 211–224.

Lamport, L., R. Shostak, and M. Pease (1982). "The Byzantine Generals Problem". In: *ACM Transactions on Programming Languages and Systems* 4.3, pp. 382–401.

Lau, H., I. Bate, P. Cairns, and J. Timmis (2011). "Adaptive data-driven error detection in swarm robotics with statistical classifiers". In: *Robotics and Autonomous Systems* 59.12, pp. 1021–1035.

LeBlanc, H. J., H. Zhang, S. Sundaram, and X. Koutsoukos (2012). "Consensus of multi-agent networks in the presence of adversaries using only local information". In: *Proceedings of the 1st international conference on High Confidence Networked Systems*, pp. 1–10.

LeBlanc, H. J. and X. D. Koutsoukos (2011). "Consensus in networked multi-agent systems with adversaries". In: *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control*. ACM Press, pp. 281–290.

LeBlanc, H. J. and X. D. Koutsoukos (2012). "Low complexity resilient consensus in networked multi-agent systems with adversaries". In: *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control*. ACM Press, pp. 5–14.

LeBlanc, H. J., H. Zhang, X. Koutsoukos, and S. Sundaram (2013). "Resilient asymptotic consensus in robust networks". In: *IEEE Journal on Selected Areas in Communications* 31.4, pp. 766–781.

Lee, S., E. Milner, and S. Hauert (2022). "A data-driven method for metric extraction to detect faults in robot swarms". In: *IEEE Robotics and Automation Letters* 7.4, pp. 10746–10753.

Lesaege, C., F. Ast, and W. George (2019). *Kleros: Short Paper*. Version v1.0.7. URL: https://kleros.io/static/whitepaper_en-8bd3a0480b45c39899787e17049ded26.pdf (visited on 11/11/2025).

Lightning developers (2025). *BOLT — Basis of Lightning Technology (protocol specifications)*. Version Commit 1560eac. URL: https://github.com/lightning/bolts (visited on 11/11/2025).

Lin, J., A. S. Morse, and B. D. Anderson (2003). "The multi-agent rendezvous problem". In: *42nd ieee international conference on decision and control*. IEEE Press, pp. 1508–1513.

Lin, Z., T. Wang, L. Shi, S. Zhang, and B. Cao (2025). "Decentralized physical infrastructure networks (DePIN): Challenges and opportunities". In: *IEEE Network* 39.2, pp. 91–99.

Liu, Y. and K. M. Passino (2004). "Stable social foraging swarms in a noisy environment". In: *IEEE Transactions on automatic control* 49.1, pp. 30–44.

Loi, A., L. Macabre, J. Fersula, K. Amini, L. Cazenille, F. Caura, A. Guerre, S. Gourichon, O. Dauchot, and N. Bredeche (2025). *Pobogot–An Open-Hardware Open-Source Low Cost Robot for Swarm Robotics*. arXiv preprint: 2504.08686 (cs.RO). URL: https://arxiv.org/abs/2504.08686.

Lokhava, M., G. Losa, D. Mazières, G. Hoare, N. Barry, E. Gafni, J. Jove, R. Malinowsky, and J. McCaleb (2019). "Fast and secure global payments with stellar". In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. ACM Press, pp. 80–96.

Lopes, V. and L. A. Alexandre (2019). "Detecting robotic anomalies using robotchain". In: *IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC 2019)*, pp. 1–6.

Lopes, V., N. Pereira, M. Fernandes, and L. A. Alexandre (2021). "A Time-Segmented Consortium Blockchain for Robotic Event Registration". In: *Proceedings of the 3rd International Conference on Blockchain Technology (ICBCT 2021)*. ACM Press, pp. 117–122.

Lourakis, M. and A. Argyros (2009). "SBA: A Software Package for Generic Sparse Bundle Adjustment". In: *ACM Transactions on Mathematical Software* 36.1, pp. 1–30.

Lustenberger, M., F. Spychiger, L. Küng, E. Bassi, and S. Wollenschläger (2025). "DAO Research Trends: Reflections and Learnings from the First European DAO Workshop (DAWO)". In: *Applied Sciences* 15.7. ISSN: 2076-3417.

Lynch, N. A. (1996). *Distributed Algorithms*. First. The Morgan Kaufmann Series in Data Management Systems. San Francisco, CA, USA: Morgan Kaufmann. ISBN: 978-0-08-050470-4.

Majcherczyk, N. and C. Pinciroli (2020). "SwarmMesh: A Distributed Data Structure for Cooperative Multi-Robot Applications". In: *IEEE International Conference on Robotics and Automation – ICRA 2020*. IEEE Press, pp. 4059–4065.

Majcherczyk, N., N. Srishankar, and C. Pinciroli (2021). "Flow-FL: Data-Driven Federated Learning for Spatio-Temporal Predictions in Multi-Robot Systems". In: *IEEE International Conference on Robotics and Automation – ICRA 2021*. IEEE Press, pp. 8836–8842.

Mallikarachchi, S., C. Dai, O. Seneviratne, and I. Godage (2022). "Managing Collaborative Tasks within Heterogeneous Robotic Swarms using Swarm Contracts". In: *Proceedings of the 4th IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS 2022)*, pp. 48–55.

Masi, G. D., J. Prasetyo, R. Zakir, N. Mankovskii, E. Ferrante, and E. Tuci (2021). "Robot swarm democracy: the importance of informed individuals against zealots". In: *Swarm Intelligence* 15.4, pp. 315–338.

Mathews, N., A. L. Christensen, R. O'Grady, F. Mondada, and M. Dorigo (2017). "Mergeable Nervous Systems for Robots". In: *Nature Communications* 8.439.

Mathews, N., A. L. Christensen, A. Stranieri, A. Scheidler, and M. Dorigo (2019). "Supervised morphogenesis: Exploiting morphological flexibility of self-assembling multirobot systems through cooperation with aerial robots". In: *Robotics and Autonomous Systems* 112, pp. 154–167.

McMahan, B., E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas (2017). "Communication-efficient learning of deep networks from decentralized data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS 2017)*. 54, pp. 1273–1282.

Merkel, D. (2014). "Docker: Lightweight Linux Containers for Consistent Development and Deployment". In: *Linux J.* 2014.2, p. 2.

Miletitch, R., V. Trianni, A. Campo, and M. Dorigo (2013). "Information Aggregation Mechanisms in Social Odometry". In: *Advances in Artificial Life, ECAL 2013: 12th European Conference on the Synthesis and Simulation of Living Systems*. MIT Press, pp. 102–109.

Millard, A., R. Joyce, J. Hilder, C. Fleşeriu, L. Newbrook, W. Li, L. McDaid, and D. Halliday (2017). "The Pi-puck extension board: A Raspberry Pi interface for the e-puck robot platform". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS 2017*. IEEE Press, pp. 741–748.

Mokhtar, A., N. Murphy, and J. Bruton (2019). "Blockchain-based Multi-Robot Path Planning". In: *Proceedings of the 5th IEEE World Forum on Internet of Things (WF–IoT 2019)*. IEEE Press, pp. 584–589.

Mondada, F., G. C. Pettinaro, A. Guignard, I. W. Kwee, D. Floreano, J.-L. Deneubourg, S. Nolfi, L. M. Gambardella, and M. Dorigo (2004). "Swarm-Bot: A New Distributed Robotic Concept". In: *Autonomous Robots* 17.2–3, pp. 193–221.

Mondada, F., M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli (2009). "The e-puck, a robot designed for education in engineering". In: *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*. Instituto Politécnico de Castelo Branco, pp. 59–65.

Monte, G. D., D. Pennino, and M. Pizzonia (2020). "Scaling blockchains without giving up decentralization and security: A solution to the blockchain scalability trilemma". In: *Proceedings of the 3rd Workshop on Cryptocurrencies and Blockchains for Distributed Systems*, pp. 71–76.

Montes de Oca, M. A., E. Ferrante, A. Scheidler, C. Pinciroli, M. Birattari, and M. Dorigo (2011). "Majority-Rule Opinion Dynamics with Differential Latency: A Mechanism for Self-Organized Collective Decision-Making". In: *Swarm Intelligence* 5.3–4, pp. 305–327.

Morlino, G., V. Trianni, and E. Tuci (2010). "Collective perception in a swarm of autonomous robots". In: *International Conference on Evolutionary Computation*, pp. 51–59.

Moroncelli, A., A. Pacheco, V. Strobel, P.-Y. Lajoie, M. Dorigo, and A. Reina (2024). "Byzantine Fault Detection in Swarm-SLAM Using Blockchain and Geometric Constraints". In: *Swarm Intelligence – Proceedings of ANTS 2024 – 14th International Conference*. Vol. 14987. Lecture Notes in Computer Science. Springer, pp. 42–56.

Mühlberger, R., S. Bachhofner, E. Castelló Ferrer, C. Di Ciccio, I. Weber, M. Wöhrer, and U. Zdun (2020). "Foundational Oracle Patterns: Connecting Blockchain to the Off-Chain World". In: *Business Process Management: Blockchain and Robotic Process Automation Forum*. Springer, pp. 35–51.

Müller, V. C. (2025). "Ethics of Artificial Intelligence and Robotics". In: *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University. URL: `https://plato.stanford.edu/archives/fall2025/entries/ethics-ai/`.

Multerer, T., A. Ganis, U. Prechtel, E. Miralles, A. Meusling, J. Mietzner, M. Vossiek, M. Loghi, and V. Ziegler (2017). "Low-cost jamming system against small drones using a 3D MIMO radar based tracking". In: *2017 European Radar Conference (EURAD)*, pp. 299–302.

Mur-Artal, R., J. Montiel, and J. Tardos (2015). "ORB-SLAM: A versatile and accurate monocular SLAM system". In: *IEEE Transactions on Robotics* 31.5, pp. 1147–1163.

Na, S., T. Rouček, J. Ulrich, J. Pikman, T. Krajník, B. Lennox, and F. Arvin (2023). "Federated Reinforcement Learning for Collective Navigation of Robotic Swarms". In: *IEEE Transactions on Cognitive and Developmental Systems* 15.4, pp. 2122–2131.

Nakamoto, S. (2008). *Bitcoin: A peer-to-peer electronic cash system*. URL: `https://bitcoin.org/bitcoin.pdf` (visited on 11/11/2025).

Nerurkar, E. D., S. I. Roumeliotis, and A. Martinelli (2009). "Distributed maximum a posteriori estimation for multi-robot cooperative localization". In: *IEEE International Conference on Robotics and Automation – ICRA 2009*, pp. 1402–1409.

Newcombe, R. A., S. J. Lovegrove, and A. J. Davison (2011). "DTAM: Dense tracking and mapping in real-time". In: *2011 International Conference on Computer Vision*, pp. 2320–2327.

Nguyen, Q. and K. Sreenath (2022). "Robust Safety-Critical Control for Dynamic Robotics". In: *IEEE Transactions on Automatic Control* 67.3, pp. 1073–1088.

Nishida, Y., K. Kaneko, S. Sharma, and K. Sakurai (2018). "Suppressing Chain Size of Blockchain-Based Information Sharing for Swarm Robotic Systems". In: *Proceedings of the Sixth International Symposium on Computing and Networking Workshops (CANDARW 2018)*. IEEE Press, pp. 524–528.

Nolfi, S. and D. Floreano (2000). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. First. Cambridge, MA, USA: MIT Press.

Nouyan, S., R. Groß, M. Bonani, F. Mondada, and M. Dorigo (2009). "Teamwork in Self-Organized Robot Colonies". In: *IEEE Transactions on Evolutionary Computation* 13.4, pp. 695–711.

Oğuz, S., E. Garone, M. Dorigo, and M. K. Heinrich (2025). *Proactive-reactive detection and mitigation of intermittent faults in robot swarms*. arXiv preprint: `2509.19246` (cs.RO). URL: `https://arxiv.org/abs/2509.19246`.

Olfati-Saber, R., J. A. Fax, and R. M. Murray (2007). "Consensus and cooperation in networked multi-agent systems". In: *Proceedings of the IEEE* 95.1, pp. 215–233.

Olfati-Saber, R. and R. M. Murray (2004). "Consensus problems in networks of agents with switching topology and time-delays". In: *IEEE Transactions on Automatic Control* 49.9, pp. 1520–1533.

Ongaro, D. and J. Ousterhout (2014). "In search of an understandable consensus algorithm". In: *2014 USENIX annual technical conference (USENIX ATC 14)*, pp. 305–319.

Pacheco, A., S. De Vos, A. Reina, M. Dorigo, and V. Strobel (2024a). "Securing Federated Learning in Robot Swarms using Blockchain Technology". In: *Proceedings of the 17th International Symposium on Distributed Autonomous Robotic Systems*. Vol. 34. Springer Proceedings in Advanced Robotics. Springer, pp. 473–488.

Pacheco, A., V. Strobel, and M. Dorigo (2020). "A Blockchain-Controlled Physical Robot Swarm Communicating via an Ad-Hoc Network". In: *Swarm Intelligence – Proceedings of ANTS 2020 – 12th International Conference*. Vol. 12421. Lecture Notes in Computer Science. Springer, pp. 3–15.

Pacheco, A., V. Strobel, A. Reina, and M. Dorigo (2022). "Real-time Coordination of a Foraging Robot Swarm using Blockchain Smart Contracts". In: *Swarm Intelligence – Proceedings of ANTS 2022 – 13th International Conference*. Vol. 13491. Lecture Notes in Computer Science. Springer, pp. 196–208.

Pacheco, A., S. De Vos, A. Reina, M. Dorigo, and V. Strobel (2024b). *Interface `geth-argos`: Trajectory Prediction*. URL: `https://zenodo.org/records/13622916` (visited on 11/11/2025).

Pacheco, A., U. Denis, R. Zakir, V. Strobel, A. Reina, and M. Dorigo (2024c). *Toychain: A Simple Blockchain for Research in Swarm Robotics*. arXiv: `2407.06630 [cs.RO]`. URL: `https://arxiv.org/abs/2407.06630`.

Pacheco, A. and V. Strobel (2020). *Interface `geth-pi-pucks`*. URL: `https://github.com/teksander/geth-pi-pucks` (visited on 11/11/2025).

Pacheco, A. and V. Strobel (2022). *Interface `geth-argos`*. URL: `https://github.com/teksander/geth-argos` (visited on 11/11/2025).

Pacheco, A., H. Zhao, V. Strobel, T. Roukny, G. Dudek, A. Reina, and M. Dorigo (2025). *Swarm Oracle: Trustless Blockchain Agreements through Robot Swarms*. arXiv preprint: `2509.15956` (cs.RO). URL: `https://arxiv.org/abs/2509.15956`.

Pais, D., P. M. Hogan, T. Schlegel, N. R. Franks, N. E. Leonard, and J. A. Marshall (2013). "A mechanism for value-sensitive decision-making". In: *PloS one* 8.9, e73216.

Parisi, C., A. Mazza, N. Pozzolini, G. Wood, and A. M. Antonopoulos (2025). *Mastering Ethereum: Implementing Smart Contracts*. Second. Sebastopol, CA, USA: O'Reilly Media. ISBN: 9781098168414.

Park, H. and S. Hutchinson (2017). "Fault-Tolerant Rendezvous of Multirobot Systems". In: *IEEE Transactions on Robotics* 33.3, pp. 565–582.

Parlangeli, G. (2013). "A fault compensation strategy for consensus networks subject to transient and intermittent faults". In: *Proceedings of the 21st Mediterranean Conference on Control and Automation*, pp. 46–53.

Peleg, D. (2005). "Distributed coordination algorithms for mobile robot swarms: New directions and challenges". In: *International Workshop on Distributed Computing*, pp. 1–12.

Peña Queralta, J., F. Keramat, S. Salimi, L. Fu, X. Yu, and T. Westerlund (2023). "Blockchain and Emerging Distributed Ledger Technologies for Decentralized Multi-robot Systems". In: *Current Robotics Reports* 4.3, pp. 43–54.

Peña Queralta, J. and T. Westerlund (2021). "Blockchain for Mobile Edge Computing: Consensus Mechanisms and Scalability". In: *Mobile Edge Computing*. Springer, pp. 333–357.

Pham, D., S. Dimov, and C. Nguyen (2004). "An incremental K-means algorithm". In: *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 218.7, pp. 783–795.

Pierro, G. A. and R. Tonelli (2022). "Can solana be the solution to the blockchain scalability problem?" In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 1219–1226.

Pinciroli, C., 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 (2012). "ARGoS: A Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems". In: *Swarm Intelligence* 6.4, pp. 271–295.

Pinciroli, C. and G. Beltrame (2016). "Buzz: A Programming Language for Robot Swarms". In: *IEEE Software* 33.4, pp. 97–100.

Pinciroli, C., A. Lee-Brown, and G. Beltrame (2016). "A Tuple Space for Data Sharing in Robot Swarms". In: *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (BICT 2015)*. ICST, pp. 287–294.

Pitonakova, L., R. Crowder, and S. Bullock (2014). "Understanding the Role of Recruitment in Collective Robot Foraging". In: *Proceedings of the 14th International Conference on the Synthesis and Simulation of Living Systems (ALIFE 14)*. MIT Press, pp. 264–271.

Pitonakova, L., R. Crowder, and S. Bullock (2018). "The Information-Cost-Reward framework for understanding robot swarm foraging". In: *Swarm Intelligence* 12, pp. 71–96.

Placed, J. A., J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos (2023). "A Survey on Active Simultaneous Localization and Mapping: State of the Art and New Frontiers". In: *IEEE Transactions on Robotics* 39.3, pp. 1686–1705.

Pluhacek, M., S. Garnier, and A. Reina (2025). "Decentralised construction of a global coordinate system in a large swarm of minimalistic robots". In: *Swarm Intelligence* 19.3, pp. 333–360.

Poon, J. and T. Dryja (2016). *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. Version v0.5.9.2. URL: https://lightning.network/lightning-network-paper.pdf (visited on 11/11/2025).

Pop, C., T. Cioara, M. Antal, I. Anghel, I. Salomie, and M. Bertoncini (2018). "Blockchain based decentralized management of demand response programs in smart energy grids". In: *Sensors* 18.1, p. 162.

Prasetyo, J., G. De Masi, P. Ranjan, and E. Ferrante (2018). "The Best-of-n Problem with Dynamic Site Qualities: Achieving Adaptability with Stubborn Individuals". In: *Swarm Intelligence – Proceedings of ANTS 2018 – 11th International Conference*. Vol. 11172. Lecture Notes in Computer Science. Springer, pp. 239–251.

Prencipe, G. (2005). "On the feasibility of gathering by autonomous mobile robots". In: *International Colloquium on Structural Information and Communication Complexity*, pp. 246–261.

Prorok, A., M. Malencia, L. Carlone, G. S. Sukhatme, B. M. Sadler, and V. Kumar (2021). *Beyond robustness: A taxonomy of approaches towards resilient multi-robot systems*. arXiv: 2109.12343 [cs.RO]. URL: https://arxiv.org/abs/2109.12343.

Raoufi, M., P. Romanczuk, and H. Hamann (2023). "Individuality in swarm robots with the case study of Kilobots: Noise, bug, or feature?" In: *ALIFE 2023: The 2023 Conference on Artificial Life*. The International Society for Artificial Life, pp. 35–45.

Reina, A., M. Dorigo, and V. Trianni (2014a). "Collective Decision Making in Distributed Systems Inspired by Honeybees Behaviour". In: *Proceedings of the 13th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2014)*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, pp. 1421–1422.

Reina, A., M. Dorigo, and V. Trianni (2014b). "Towards a Cognitive Design Pattern for Collective Decision-Making". In: *Swarm Intelligence – Proceedings of ANTS 2014 – 9th International Conference*. Vol. 8667. Lecture Notes in Computer Science. Springer, pp. 194–205.

Reina, A., G. Valentini, C. Fernández-Oto, M. Dorigo, and V. Trianni (2015). "A Design Pattern for Decentralised Decision Making". In: *PLOS ONE* 10.10, e0140950.

Reina, A. (2020). "Robot teams stay safe with blockchains". In: *Nature Machine Intelligence* 2.5, pp. 240–241.

Reina, A., R. Zakir, G. De Masi, and E. Ferrante (2023). "Cross-inhibition leads to group consensus despite the presence of strongly opinionated minorities and asocial behaviour". In: *Communications Physics* 6.1, p. 236.

Rocket, T., M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer (2019). *Scalable and probabilistic leaderless BFT consensus through metastability*. arXiv: 1906.08936 [cs.DC]. URL: https://arxiv.org/abs/1906.08936.

Rodriguez-Losada, D., F. Matia, and A. Jimenez (2004). "Local maps fusion for real time multirobot indoor simultaneous localization and mapping". In: *Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 2, pp. 1308–1313.

Roşu, I. and F. Saleh (2021). "Evolution of shares in a proof-of-stake cryptocurrency". In: *Management Science* 67.2, pp. 661–672.

Roumeliotis, S. I., G. S. Sukhatme, and G. A. Bekey (1998). "Sensor fault detection and identification in a mobile robot". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS 1998*. Vol. 3, pp. 1383–1388.

Roy, N. and G. Dudek (2001). "Collaborative robot exploration and rendezvous: Algorithms, performance bounds and observations". In: *Autonomous Robots* 11.2, pp. 117–136.

Rubenstein, M., C. Ahler, and R. Nagpal (2012). "Kilobot: A low cost scalable robot system for collective behaviors". In: *IEEE International Conference on Robotics and Automation – ICRA 2012*, pp. 3293–3298.

Sabater, J. and C. Sierra (2005). "Review on Computational Trust and Reputation Models". In: *Artificial Intelligence Review* 24.1, pp. 33–60.

Saeedi, S., M. Trentini, M. Seto, and H. Li (2016a). "Multiple-Robot Simultaneous Localization and Mapping: A Review". In: *Journal of Field Robotics* 33, pp. 3–46.

Saeedi, S., M. Trentini, M. Seto, and H. Li (2016b). "Multiple-robot simultaneous localization and mapping: A review". In: *Journal of Field Robotics* 33.1, pp. 3–46.

Şahin, E. (2005). "Swarm Robotics: From Sources of Inspiration to Domains of Application". In: *Swarm Robotics*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 10–20.

Salah, K., M. H. U. Rehman, N. Nizamuddin, and A. Al-Fuqaha (2019). "Blockchain for AI: Review and open research challenges". In: *IEEE Access* 7, pp. 10127–10149.

Saleh, F. (2021). "Blockchain without waste: Proof-of-stake". In: *The Review of financial studies* 34.3, pp. 1156–1190.

Salimi, S., J. Peña Queralta, and T. Westerlund (2023). "Hyperledger Fabric Blockchain and ROS 2 Integration for Autonomous Mobile Robots". In: *2023 IEEE/SICE International Symposium on System Integration*. IEEE Press, pp. 1–8.

Salimpour, S., F. Keramat, J. P. Queralta, and T. Westerlund (2023). "Decentralized Vision-Based Byzantine Agent Detection In Multi-Robot Systems With IOTA

Smart Contracts". In: *Foundations and Practice of Security: 15th International Symposium, FPS 2022, Revised Selected Papers*. Springer, pp. 322–337.

Salman, M., D. Garzón Ramos, K. Hasselmann, and M. Birattari (2020). "Phormica: Photochromic pheromone release and detection system for stigmergic coordination in robot swarms". In: *Frontiers in Robotics and AI* 7, p. 591402.

Santos De Campos, M. G., C. P. Chanel, C. Chauffaut, and J. Lacan (2021). "Towards a Blockchain-based Multi-UAV Surveillance System". In: *Frontiers in Robotics and AI* 8, p. 557692.

Saulnier, K., D. Saldaña, A. Prorok, G. J. Pappas, and V. Kumar (2017). "Resilient Flocking for Mobile Robot Teams". In: *IEEE Robotics and Automation Letters* 2.2, pp. 1039–1046.

Schmuck, P., T. Ziegler, M. Karrer, J. Perraudin, and M. Chli (2021). "COVINS: Visual-Inertial SLAM for Centralized Collaboration". In: *2021 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. Los Alamitos, CA: IEEE Computer Society Press, pp. 171–176.

Seeley, T. D. (1983). "Division of Labor between Scouts and Recruits in Honeybee Foraging". In: *Behavioral Ecology and Sociobiology* 12, pp. 253–259.

Shifferaw, Y. and S. Lemma (2021). "Limitations of proof of stake algorithm in blockchain: A review". In: *Zede Journal* 39.1, pp. 81–95.

Shoham, Y. and K. Leyton-Brown (2008). *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge, UK: Cambridge University Press. ISBN: 978-0-511-81165-4.

Silencio Network (2025). *Silencio whitepaper*. URL: `https://whitepaper.silencio.network/` (visited on 11/11/2025).

Simoens, P., M. Dragone, and A. Saffiotti (2018). "The Internet of Robotic Things: A review of concept, added value and applications". In: *International journal of advanced robotic systems* 15.1, p. 1729881418759424.

Sperati, V., V. Trianni, and S. Nolfi (2011). "Self-organised path formation in a swarm of robots". In: *Swarm Intelligence* 5, pp. 97–119.

Stephens, D. W. and J. R. Krebs (1986). *Foraging Theory*. Princeton, NJ, USA: Princeton University Press. ISBN: 978-0-691-08442-8.

Strobel, V., E. Castelló Ferrer, and M. Dorigo (2018). "Managing Byzantine Robots via Blockchain Technology in a Swarm Robotics Collective Decision Making Scenario". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2018)*. AAMAS '18. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, pp. 541–549.

Strobel, V., E. Castelló Ferrer, and M. Dorigo (2020). "Blockchain technology secures robot swarms: A comparison of consensus protocols and their resilience to Byzantine robots". In: *Frontiers in Robotics and AI* 7.54.

Strobel, V. and M. Dorigo (2018). "Blockchain Technology for Robot Swarms: A Shared Knowledge and Reputation Management System for Collective Estimation". In: *Swarm Intelligence – Proceedings of ANTS 2018 – 11th International Conference*. Vol. 11172. Lecture Notes in Computer Science. Springer, pp. 425–426.

Strobel, V., A. Pacheco, and M. Dorigo (2023). "Robot Swarms Neutralize Harmful Byzantine Robots Using a Blockchain-based Token Economy". In: *Science Robotics* 8.79, eabm4636.

Sundaram, S. and C. N. Hadjicostis (2008a). "Distributed function calculation via linear iterations in the presence of malicious agents–Part I: Attacking the network". In: *Proceedings of the American Control Conference*, pp. 1350–1355.

Sundaram, S. and C. N. Hadjicostis (2008b). "Distributed function calculation via linear iterations in the presence of malicious agents–Part II: Overcoming malicious behavior". In: *Proceedings of the American Control Conference*, pp. 1356–1361.

Sundaram, S., S. Revzen, and G. J. Pappas (2012). "A control-theoretic approach to disseminating values and overcoming malicious links in wireless networks". In: *Automatica* 48.11, pp. 2894–2901.

Suzuki, I. and M. Yamashita (1999). "Distributed Anonymous Mobile Robots: Formation of Geometric Patterns". In: *SIAM Journal on Computing* 28.4, pp. 1347–1363.

Szabo, N. (1997). "Formalizing and securing relationships on public networks". In: *First Monday* 2.9.

Szilágyi, P. (2017). *EIP-225: Clique proof-of-authority consensus protocol*. URL: `https://eips.ethereum.org/EIPS/eip-225` (visited on 11/11/2025).

Talamali, M. S., T. Bose, M. Haire, X. Xu, J. A. R. Marshall, and A. Reina (2020). "Sophisticated collective foraging with minimalist agents: A swarm robotics test". In: *Swarm Intelligence* 14.1, pp. 25–56.

Talamali, M. S., J. A. Marshall, T. Bose, and A. Reina (2019). "Improving collective decision accuracy via time-varying cross-inhibition". In: *IEEE International Conference on Robotics and Automation – ICRA 2019*. IEEE Press, pp. 9652–9659.

Talamali, M. S., A. Saha, J. A. Marshall, and A. Reina (2021). "When less is more: Robot swarms adapt better to changes with constrained communication". In: *Science Robotics* 6.56, eabf1416.

Tang, Y., Y. Lv, J. Zhou, and M. Ogorzałek (2025). "Resilient Consensus in Open Multi-Agent Systems". In: *IEEE Control Systems Letters* 9, pp. 1261–1266.

Tarapore, D., A. L. Christensen, and J. Timmis (2017). "Generic, scalable and decentralized fault detection for robot swarms". In: *PLOS ONE* 12.8, pp. 1–29.

Tarapore, D., P. U. Lima, J. Carneiro, and A. L. Christensen (2015). "To err is robotic, to tolerate immunological: fault detection in multirobot systems". In: *Bioinspiration & Biomimetics* 10.1, p. 016014.

Tarapore, D., J. Timmis, and A. L. Christensen (2019). "Fault Detection in a Swarm of Physical Robots Based on Behavioral Outlier Detection". In: *IEEE Transactions on Robotics* 35.6, pp. 1516–1522.

Tellor (2023). *Tellor whitepaper*. URL: `https://tellor.io/whitepaper/` (visited on 11/11/2025).

Thakur, A., S. Sahoo, A. Mukherjee, and R. Halder (2023). "Making Robotic Swarms Trustful: A Blockchain-Based Perspective". In: *Journal of Computing and Information Science in Engineering* 23.6, p. 060803.

Tian, Y., Y. Chang, F. Herrera Arias, C. Nieto-Granda, J. P. How, and L. Carlone (2022). "Kimera-Multi: Robust, Distributed, Dense Metric-Semantic SLAM for Multi-Robot Systems". In: *IEEE Transactions on Robotics* 38.4, pp. 2022–2038.

Timmis, J., A. Ismail, J. Bjerknes, and A. Winfield (2016). "An immune-inspired swarm aggregation algorithm for self-healing swarm robotic systems". In: *Biosystems* 146, pp. 60–76. ISSN: 0303-2647.

Tran, J. A., G. S. Ramachandran, P. M. Shah, C. B. Danilov, R. A. Santiago, and B. Krishnamachari (2019). "SwarmDAG: A Partition Tolerant Distributed Ledger Protocol for Swarm Robotics". In: *Ledger* 4.

Trianni, V., S. Nolfi, and M. Dorigo (2008). "Evolution, Self-organization and Swarm Robotics". In: *Swarm Intelligence: Introduction and Applications*. Natural Computing Series. Amsterdam, The Netherlands: Springer. Chap. 5, pp. 43–85.

Trianni, V. and A. Campo (2015). "Fundamental collective behaviors in swarm robotics". In: *Springer handbook of computational intelligence*. Springer, pp. 1377–1394.

Turing, A. M. (1939). "Systems of logic based on ordinals". In: *Proceedings of the London Mathematical Society, Series 2* 45, pp. 161–228.

UMA Optimistic Oracle (2025). *UMA Optimistic Oracle (protocol documentation and whitepaper)*. URL: `https://docs.uma.xyz/` (visited on 11/11/2025).

Valentini, G., A. Antoun, M. Trabattoni, B. Wiandt, Y. Tamura, E. Hocquard, V. Trianni, and M. Dorigo (2018). "Kilogrid: A Novel Experimental Environment for the Kilobot Robot". In: *Swarm Intelligence* 12.3, pp. 245–266.

Valentini, G., D. Brambilla, H. Hamann, and M. Dorigo (2016). "Collective Perception of Environmental Features in a Robot Swarm". In: *Swarm Intelligence – Proceedings of ANTS 2016 – 10th International Conference*. Vol. 9882. Lecture Notes in Computer Science. Springer, pp. 65–76.

Valentini, G., E. Ferrante, and M. Dorigo (2017). "The Best-of-n Problem in Robot Swarms: Formalization, State of the Art, and Novel Perspectives". In: *Frontiers in Robotics and AI* 4.9.

Valentini, G., H. Hamann, and M. Dorigo (2015a). "Efficient Decision-Making in a Self-Organizing Robot Swarm: On the Speed Versus Accuracy Trade-Off". In: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*. IFAAMAS, pp. 1305–1314.

Valentini, G., H. Hamann, and M. Dorigo (2015b). "Self-organized collective decision-making in a 100-robot swarm". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*. AAAI press, pp. 4216–4217.

Valentini, G., H. Hamann, and M. Dorigo (2015c). "Self-organized Collective Decisions in a Robot Swarm". In: *AAAI-15 Video Proceedings*. AAAI Press.

Van Calck, L., A. Pacheco, V. Strobel, M. Dorigo, and A. Reina (2023). "A blockchain-based information market to incentivise cooperation in swarms of self-interested robots". In: *Scientific Reports* 13.20417.

Varadharajan, V. S., K. Soma, S. Dyanatkar, P.-Y. Lajoie, and G. Beltrame (2024). *Hierarchies define the scalability of robot swarms*. arXiv preprint: `2405.02417` (cs.RO). URL: `https://arxiv.org/abs/2405.02417`.

Vicsek, T., A. Czirók, E. Ben-Jacob, I. Cohen, and O. Shochet (1995). "Novel Type of Phase Transition in a System of Self-Driven Particles". In: *Physical Review Letters* 75.6, pp. 1226–1229.

Wang, S., W. Ding, J. Li, Y. Yuan, L. Ouyang, and F.-Y. Wang (2019). "Decentralized Autonomous Organizations: Concept, Model, and Applications". In: *IEEE Transactions on Computational Social Systems* 6.5, pp. 870–878.

Wardega, K., M. von Hippel, R. Tron, C. Nita-Rotaru, and W. Li (2023). "Byzantine Resilience at Swarm Scale: A Decentralized Blocklist Protocol from Inter-Robot Accusations". In: *Proceedings of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023)*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 1430–1438.

Warnat-Herresthal, S., H. Schultze, K. L. Shastry, S. Manamohan, S. Mukherjee, V. Garg, R. Sarveswara, K. Händler, P. Pickkers, N. A. Aziz, et al. (2021). "Swarm learning for decentralized and confidential clinical machine learning". In: *Nature* 594.7862, pp. 265–270.

WeatherXM (2025). *WeatherXM documentation*. Version v0.6.2. URL: `https://docs.weatherxm.com/` (visited on 11/11/2025).

Wen, J., Z. Zhang, Y. Lan, Z. Cui, J. Cai, and W. Zhang (2023). "A survey on federated learning: challenges and applications". In: *International journal of machine learning and cybernetics* 14.2, pp. 513–535.

White, R., G. Caiazza, A. Cortesi, Y. Cho, and H. Christensen (2019). "Black Block Recorder: Immutable Black Box Logging for Robots via Blockchain". In: *IEEE Journal on Robotics and Automation* 4, pp. 3812–3819.

Wilson, E. O. (2000). *Sociobiology: The New Synthesis*. Twenty-Fifth Anniversary Edition. Cambridge, MA, USA: Harvard University Press. ISBN: 978-0-674-00235-7.

Wilson, J., G. Chance, P. Winter, S. Lee, E. Milner, D. Abeywickrama, S. Windsor, J. Downer, K. Eder, J. Ives, and S. Hauert (2023). "Trustworthy Swarms". In: *Proceedings of the First International Symposium on Trustworthy Autonomous Systems*. ACM Press.

Winfield, A. F. T. and J. Nembrini (2006). "Safety in numbers: Fault tolerance in robot swarms". In: *International Journal on Modelling Identification and Control* 1.1, pp. 30–37.

Winfield, A. F. (2009). "Foraging Robots". In: *Encyclopedia of Complexity and Systems Science*. New York, NY: Springer, pp. 3682–3700. ISBN: 978-0-387-30440-3.

Xianjia, Y., J. P. Queralta, J. Heikkonen, and T. Westerlund (2021). "Federated learning in robotic and autonomous systems". In: *Procedia Computer Science* 191, pp. 135–142.

Xu, G., H. Li, S. Liu, K. Yang, and X. Lin (2019). "Verifynet: Secure and verifiable federated learning". In: *IEEE Transactions on Information Forensics and Security* 15, pp. 911–926.

Xu, Y., G. Deng, T. Zhang, H. Qiu, and Y. Bao (2021). "Novel denial-of-service attacks against cloud-based multi-robot systems". In: *Information Sciences* 576, pp. 329–344.

Yakovenko, A. (2018). *Solana: A new architecture for a high performance blockchain*. Version v0.8.13. URL: `https://solana.com/solana-whitepaper.pdf` (visited on 11/11/2025).

Yang, G.-Z., J. Bellingham, P. E. Dupont, P. Fischer, L. Floridi, R. Full, N. Jacobstein, V. Kumar, M. McNutt, R. Merrifield, B. J. Nelson, B. Scassellati, M. Taddeo, R. Taylor, M. Veloso, Z. L. Wang, and R. Wood (2018). "The grand challenges of Science Robotics". In: *Science Robotics* 3.14, eaar7650.

Yang, R., F. R. Yu, P. Si, Z. Yang, and Y. Zhang (2019). "Integrated blockchain and edge computing systems: A survey, some research issues and challenges". In: *IEEE Communications Surveys & Tutorials* 21.2, pp. 1508–1532.

Yin, M., D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham (2019). "HotStuff: BFT consensus with linearity and responsiveness". In: *Proceedings of the 2019 ACM symposium on principles of distributed computing*, pp. 347–356.

Yu, X., D. Saldaña, D. Shishika, and M. A. Hsieh (2021). "Resilient consensus in robot swarms with periodic motion and intermittent communication". In: *IEEE Transactions on Robotics* 38.1, pp. 110–125.

Zabka, P., K. Förster, C. Decker, and S. Schmid (2023). "A centrality analysis of the Lightning Network". In: *Telecommunications Policy* 47.10, p. 102696.

Zakir, R., M. Dorigo, and A. Reina (2022). "Robot Swarms Break Decision Deadlocks in Collective Perception Through Cross-Inhibition". In: *Swarm Intelligence – Proceedings of ANTS 2022 – 13th International Conference*. Vol. 13491. Lecture Notes in Computer Science. Springer, pp. 209–221.

Zakir, R., M. Dorigo, and A. Reina (2024a). "Robot Swarms Break Decision Deadlocks in Collective Perception Through Cross-Inhibition". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS 2024*. IEEE Press, pp. 9014–9021.

Zakir, R., M. Dorigo, and A. Reina (2024b). "Miscommunication between robots can improve group accuracy in best-of-n decision-making". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS 2024*. IEEE Press, pp. 9014–9021.

Zhang, F., E. Cecchetti, K. Croman, A. Juels, and E. Shi (2016). "Town crier: An authenticated data feed for smart contracts". In: *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security*, pp. 270–282.

Zhang, H., E. Fata, and S. Sundaram (2015). "A notion of robustness in complex networks". In: *IEEE Transactions on Control of Network Systems* 2.3, pp. 310–320.

Zhang, H. and S. Sundaram (2012a). "A simple median-based resilient consensus algorithm". In: *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1734–1741.

Zhang, H. and S. Sundaram (2012b). "Robustness of information diffusion algorithms to locally bounded adversaries". In: *2012 American Control Conference (ACC)*, pp. 5855–5861.

Zhang, Y., Y. Wu, K. Tong, H. Chen, and Y. Yuan (2023). "Review of Visual Simultaneous Localization and Mapping Based on Deep Learning". In: *Remote Sensing* 15.11, p. 2740.

Zhao, H., A. Pacheco, G. Beltrame, X. Liu, M. Dorigo, and G. Dudek (2025). "A Blockchain Framework for Equitable and Secure Task Allocation in Robot Swarms". In: *IEEE Robotics and Automation Letters* 10.10, pp. 10862–10869.

Zhao, H., A. Pacheco, V. Strobel, A. Reina, X. Liu, G. Dudek, and M. Dorigo (2023). "A Generic Framework for Byzantine-tolerant Consensus Achievement in Robot Swarms". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems – IROS 2023*. IEEE Press, pp. 8839–8846.

Zhao, W., M. Ammar, and E. Zegura (2004). "A message ferrying approach for data delivery in sparse mobile ad hoc networks". In: *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pp. 187–198.

Zhong, S., Y. Qi, Z. Chen, J. Wu, H. Chen, and M. Liu (2024). "DCL-SLAM: A Distributed Collaborative LiDAR SLAM Framework for a Robotic Swarm". In: *IEEE Sensors Journal* 24.4, pp. 4786–4797.

Zhou, X., W. Liang, K. I.-K. Wang, Z. Yan, L. T. Yang, W. Wei, J. Ma, and Q. Jin (2023). "Decentralized P2P Federated Learning for Privacy-Preserving and Resilient Mobile Robotic Systems". In: *IEEE Wireless Communications* 30.2, pp. 82–89.

Zhu, W., S. Oğuz, M. K. Heinrich, M. Allwright, M. Wahby, A. L. Christensen, E. Garone, and M. Dorigo (2024). "Self-organizing Nervous Systems for Robot Swarms". In: *Science Robotics* 9.96, eadl5161.

Zlot, R., A. Stentz, M. B. Dias, and S. Thayer (2002). "Multi-robot exploration controlled by a market economy". In: *IEEE International Conference on Robotics and Automation – ICRA 2002*. IEEE Press, pp. 3016–3023.

Zlot, R. and A. Stentz (2006). "Market-based multirobot coordination for complex tasks". In: *The International Journal of Robotics Research* 25.1, pp. 73–101.