POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Protecting Collaborative SLAM through Reputation Mechanisms

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria dell'Automazione

Author: **Iacopo Fornai**

Student ID: 232967
Advisor: Prof. Francesco Amigoni
Co-advisors: Dr. Andreagiovanni Reina, Ir. Alexandre Pacheco
Academic Year: 2024-2025

Abstract

Multi-robot systems already constitute the backbone of many modern robotics applications, from warehouse management to self-driving cars, and have the potential to impact other fields, including search and rescue and planetary exploration. These applications involve a team of robots executing a coordinated task in an unknown or partially known environment and require the robots to have a shared understanding of the environment and their location within it.

SLAM (Synchronous Localization And Mapping) enables one or more robot to build a map of the environment while localizing themselves within it. After decades of studies that led to robust and efficient single-robot SLAM systems, recent attention has shifted to collaborative multi-robot SLAM, where the agents cooperate in order to achieve a consistent global map by exchanging local information. Multi-robot SLAM provides valuable opportunities for enhanced efficiency, robustness, and parallelization. However, it also poses significant challenges related to scalability and data consistency.

Recent studies showed that the robustness of a multi-robot system can be compromised in the presence of even a single misbehaving or malicious robot that shares false information with the other agents. Specifically, in a collaborative SLAM scenario, corrupted information can lead to the creation of an incorrect map.

This thesis studies the robustness of a state-of-the-art framework for multi-robot SLAM called Swarm-SLAM. Following a recent study that first proposed a protection layer on Swarm-SLAM, I show that the system is still vulnerable to the presence of certain types of misbehaving robots (which are called Byzantine robots in the literature) that can act on different sections of the framework. Inspired by recent studies on blockchain-based swarm robotics, I demonstrate that blockchain technology can enhance the robustness of a multi-robot SLAM system by enabling the rejection of incorrect information and the identification of Byzantine robots. Therefore, I create a second protection layer for swarm-SLAM developing a reputation mechanism to penalise Byzantine robots that I test with a set of simulations using a variable number of robots. Through this reputation mechanism, the accuracy of the map is improved in comparison with the solution returned

by the unprotected swarm-SLAM

This work also explores the physical implementation of swarm-SLAM, highlighting the practical challenges faced during real-world deployment, particularly hardware limitations associated with the use of low-cost robots, typical of swarm robotics.

Keywords: swarm-SLAM, C-SLAM, multi-robot-SLAM, swarm-robotics, collective decision making, blockchain technology

Abstract in lingua italiana

I sistemi multi-robot costituiscono già oggi la base di molte applicazioni robotiche moderne, dalla gestione dei magazzini ai veicoli autonomi, e hanno il potenziale per influenzare anche altri settori, come le operazioni di ricerca e soccorso o l'esplorazione planetaria. Queste applicazioni prevedono l'impiego di un gruppo di robot impegnati in un compito coordinato in un ambiente sconosciuto o parzialmente noto, e richiedono che i robot condividano una comprensione comune dell'ambiente e della propria posizione al suo interno.

La tecnica SLAM (Synchronous Localization And Mapping) consente a uno o più robot di costruire una mappa dell'ambiente e allo stesso tempo di localizzarsi al suo interno. Dopo decenni di ricerca che hanno portato allo sviluppo di sistemi SLAM per un singolo robot robusti ed efficienti, l'attenzione si è recentemente spostata sullo SLAM collaborativa multi-robot, in cui gli agenti cooperano per ottenere una mappa globale coerente attraverso lo scambio di informazioni locali. Lo SLAM multi-robot offre importanti opportunità in termini di efficienza, robustezza e parallelizzazione. Tuttavia, presenta anche sfide significative legate alla scalabilità e all'integrazione coerente di dati potenzialmente in conflitto.

Studi recenti hanno dimostrato che la robustezza dei sistemi multi-robot può essere compromessa in presenza di robot malfunzionanti o malevoli, in grado di diffondere informazioni errate agli altri agenti. In particolare, in scenari di SLAM collaborativo, informazioni corrotte possono portare alla creazione di mappe non corrette.

Questa tesi analizza la robustezza di un framework all'avanguardia per lo SLAM 3D multi-robot chiamato swarm-SLAM. A partire da un recente studio che ha introdotto un primo livello di protezione in swarm-SLAM, mostro come il sistema sia ancora vulnerabile alla presenza di uno o più robot malevoli o malfunzionanti (noti in letteratura come robot bizantini) che possono agire su diverse componenti del framework. Inoltre, ispirandomi a recenti studi sulla robotica a sciame basata su blockchain, dimostro come la tecnologia blockchain possa migliorare la robustezza di un sistema multi-robot permettendo il rifiuto di informazioni errate e l'identificazione dei robot bizantini. A tal fine, ho sviluppato un secondo livello di protezione per swarm-SLAM, basato su un meccanismo di reputazione

che penalizza i robot bizantini, testandone l'efficacia attraverso una serie di simulazioni con un numero variabile di robot. Attraverso questo sistema di reputazione l'accuratezza della mappa generata è migliore rispetto alla soluzione ottenuta senza nessun livello di protezione.

Questo lavoro esplora anche l'implementazione fisica di swarm-SLAM, mettendo in evidenza le sfide pratiche incontrate durante lo sviluppo nel mondo reale, in particolare le limitazioni hardware legate all'uso di robot a basso costo tipici della robotica a sciame.

Parole chiave: SLAM con sciami di robot, SLAM collaborativo, SLAM multi-robot, sciame di robot, sistema collettivo decisionale, tecnologia blockchain

Contents

1 | Introduction

This master's thesis was carried out in collaboration with University of Konstanz in Germany, as part of a joint research project. I had the privilege of working in the Centre for Advanced Study of Collective Behaviour under the supervision of Dr. Andreagiovanni Reina and Alexandre Pacheco.

1.1. Problem statement

This study explores the field of swarm robotics [18] and distributed decision-making [61], focusing specifically on security challenges encountered by robotic swarms performing Simultaneous Localization and Mapping (SLAM) in scenarios where some robots exhibit Byzantine behavior, i.e., arbitrary or malicious actions that can disrupt the system.

SLAM constitutes a core component of modern robotic systems, as it enables one or more robots to build a map of the environment while simultaneously localizing themselves within it. It is already used in many real-world applications, such as autonomous vehicles for urban navigation, space exploration with rovers on Mars, and search and rescue operations carried out by drones. SLAM also plays a key role in industry, where mobile robots can move autonomously in warehouses without the need for fixed or centralized control infrastructure.

My thesis focuses specifically on Collaborative SLAM (C-SLAM), where robots cooperate by exchanging information in order to map a large unknown environment more efficiently than a single robot could on its own. However, this reliance on communication introduces the need for robustness and security. A C-SLAM framework must be resilient to both perception and communication errors; otherwise, the integrity of the resulting map may be compromised. The research conducted in this thesis shows that Swarm-SLAM [40]—a state of the art framework for C-SLAM—demonstrates robustness to perception errors but still lacks protection against Byzantine robots, which can inject false information into the system.

A Byzantine robot behaves improperly or maliciously, either due to malfunctioning or

malware, thereby disrupting the normal operation of the swarm. This can be particularly critical in a C-SLAM application, as even the actions of a single Byzantine robot may result in an inaccurate map of the environment and erroneous trajectory estimates. Such issues could compromise the success of a planetary exploration mission, while in an industrial environment they could cause mobile robots to navigate incorrectly, leading to collisions, production delays, or damage to goods and infrastructure.

Swarm-SLAM [40] represents a groundbreaking decentralized SLAM approach that utilizes robot swarms to achieve scalable and flexible mapping in dynamic, unknown environments. Although promising, this area is still in its early stages, with few existing frameworks and limited results regarding security and robustness against Byzantine robots that act adversarially relative to the swarm's objectives. Building on the first Byzantine fault-tolerant protocol specifically designed for the Swarm-SLAM framework [50], this thesis extends the protection to previously unprotected components of the system. The work includes a comprehensive analysis of the potential impacts of various attack types on distributed systems and introduces novel mitigation strategies leveraging blockchain technology and reputation mechanisms.

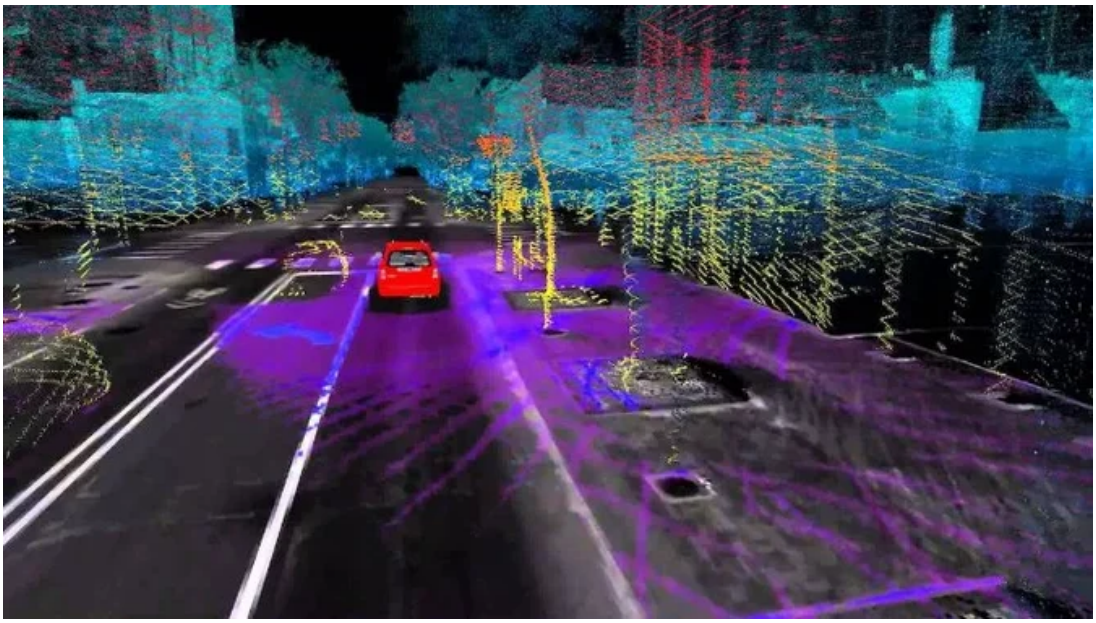

Figure 1.1: Example of a map generated by an autonomous vehicle using its onboard sensors.

1.2. Structure and methodology

The thesis is organised as follows. Chapter 1 provides a brief introduction to the motivations behind this work, explaining why SLAM is a fundamental capability in robotics and why it is crucial for it to be robust against different types of errors. Chapter 2 gives a more technical description of SLAM and of the technologies employed, and it also presents an overview of the state of the art, with particular attention to the most innovative aspects of recently proposed frameworks. Chapter 3 analyses the various vulnerabilities of the Swarm-SLAM modules when facing Byzantine robots. Chapter 4 describes an existing solution designed to secure an initial module of Swarm-SLAM and then presents my contribution aimed at protecting additional, previously unprotected modules. Chapter 5 discusses the challenges and requirements of a real-world implementation of the protected Swarm-SLAM framework. Finally, Chapter 6 summarises the key findings and offers suggestions for future research directions.

My methodology began with an in-depth study of C-SLAM systems, specifically focusing on Swarm-SLAM, to understand how these systems operate. I then explored the mechanics of blockchain technology and smart contracts, studying their application in the front-end security layer. In the SLAM context, the front-end is the module responsible for processing sensor data and building constraints for localization and mapping, and the security layer serves as a protection mechanism that ensures the integrity and reliability of this input data. Subsequently, I analyzed how Byzantine agents could compromise the system by exploiting vulnerabilities in unprotected parts, such as the back-end — the component responsible for integrating data from the front-end and performing the optimization needed for accurate localization and mapping. To address this, I developed protection mechanisms that integrate blockchain technology with a reputation system, thereby safeguarding both the front-end and back-end from malicious actors. This combined approach significantly enhances the overall robustness and security of the swarm SLAM framework. Moreover this work can be applied not only to Swarm-SLAM but also on every SLAM system that deploys Pose Graph Optimization (PGO) in the back-end.

1.3. Scientific contribution

In my thesis, I build upon existing research by studying and summarizing the theoretical robustness limitations of the most advanced framework for Collaborative SLAM. Through my analysis, I examine the impact of security faults within such systems and demonstrate how the presence of Byzantine robots in the swarm can undermine the entire system and

compromise the reliability of the final map. To address these challenges, I propose an extension aimed at enhancing Byzantine fault-tolerance specifically in the Swarm-SLAM framework [40], and more generally in any robot swarm SLAM system.

- **Contribution 1:** I extended the Byzantine fault-tolerant protocol to protect the Swarm-SLAM back-end. In particular I focused on the ways a Byzantine robot can influence the back-end and I proposed and tested a solution based on a reputation mechanism to avoid it.
- **Contribution 2:** I identified the challenges involved in transferring the protected Swarm-SLAM system from simulation to real-world deployment. This includes uncovering all the bottlenecks and hardware requirements necessary for practical implementation.
- **Contribution 3:** Based on the obtained results, I identified the most critical challenges for Byzantine-Fault-Tolerance (BFT) in Swarm-SLAM and listed potential approaches to tackle each of them

These contributions mark an important step in tackling security challenges within Collaborative SLAM systems. My thesis emphasizes the critical role of security in a future where autonomous robots extensively collaborate to perform self-localization and exploration in the absence of a central authority. In this setting, ensuring robust security guarantees is essential.

2 | Background and state of the art

In this chapter, I provide a detailed description of SLAM and C-SLAM frameworks. I also introduce blockchain technology, including the specific blockchain implementation used in this thesis. Finally, I present an overview of the state of the art, focusing on recent trends in the C-SLAM literature.

2.1. Simultaneous Localization And Mapping (SLAM)

SLAM is a key capability of autonomous systems that enables a robot or autonomous vehicle to construct a map of an unknown environment and simultaneously determine its own position within that map in real time, without relying on external localization systems such as GPS. Successfully addressing this challenge is considered a critical first step toward achieving fully autonomous operation [76].

In many robotic applications, ranging from autonomous vehicles to rescue robots operating in inaccessible environments, reliable SLAM algorithms play a crucial enabling role [21]. The practical adoption of SLAM has been steadily growing, propelled by advancements in high-quality, affordable cameras, and LiDAR sensors, along with the increasing computational power of embedded systems. A notable example of recent progress in SLAM research is active SLAM [59], where robots intelligently plan and control their movements to generate the most accurate maps possible.

The concept of SLAM was first introduced in a landmark paper by Smith and Cheeseman in 1987 [68]. Their work presented an estimation theory-based framework for robot localization and mapping, effectively bridging robotics and artificial intelligence. Over the past three decades, SLAM has experienced substantial evolution. In 1988, Ayache and Faugeras [2] pioneered research in visual navigation, and Crowley investigated Kalman filter-based visual navigation approaches for mobile robots [13]. Although early on bundle adjustment optimization techniques were less commonly applied due to their high

computational cost, the notion of sparsity in bundle adjustment emerged in 2009 [45], which enabled graph-based optimization methods such as DTAM [54] , LSD-SLAM [23] , ORB-SLAM [51] , ORB-SLAM3 [7] and other influential algorithms.

With the rapid advancements in artificial intelligence and deep learning, researchers have begun integrating deep learning with traditional filtering and graph-optimization approaches to better address challenges related to environmental adaptability faced by conventional methods [67]. As a result, deep learning applications for SLAM have become a prominent and rapidly evolving area of study.

Generally speaking, a SLAM system architecture can be viewed as consisting of two core submodules: the front end and the back end (see Figure 2.1). While these components tackle distinct problems, they operate in a continuous feedback loop, exchanging information to progressively refine the mapping and localization results. The front end is in charge of data collection, sensor data processing, feature extraction, feature matching, and estimation of robot's motion. The back end is sensor agnostic and it accomplishes data association, optimisation, map building, and consistency checking. Together, these components enable the SLAM system to provide an accurate estimate of the robot's trajectory and the map of its environment.

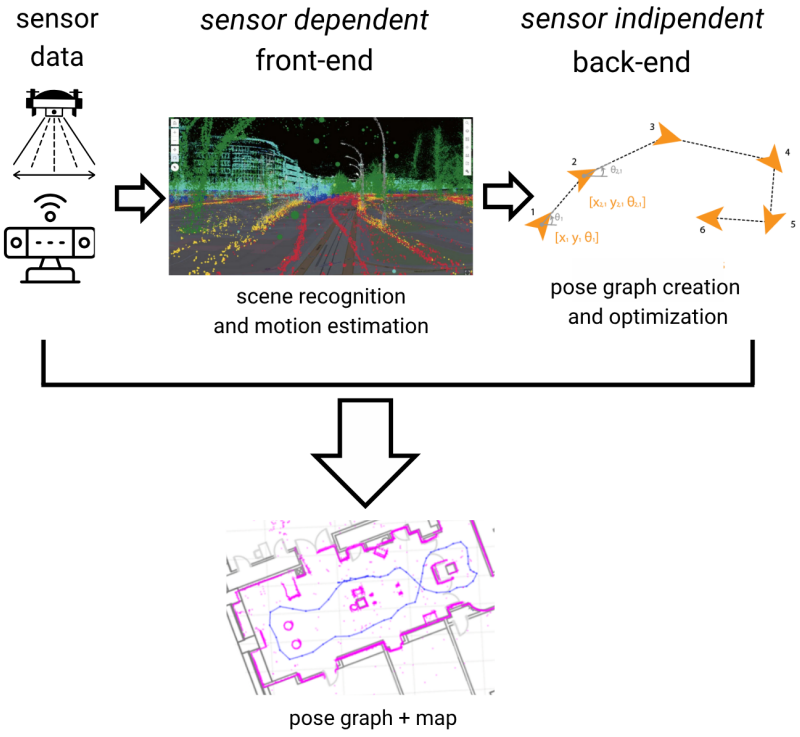

Figure 2.1: Typical SLAM information flow.

2.2. Swarm robotics

Swarm robotics is an innovative field inspired by the collective behavior of social animals. By adhering to simple rules and relying on short-range interactions, swarm robotics seeks to design and develop systems composed of numerous robots that self-organize to collaboratively accomplish tasks. The essential characteristics and properties of a robot swarm include:

- **Autonomy**: the ability to perform operations and to make decisions without continuous human intervention.
- **Scalability**: a robotic system can be composed of a large number of agents that cooperate to achieve a common goal. This ability requires advanced algorithms and communication systems to optimise the overall system performance. Scalability is a key property of such systems, allowing them to efficiently adapt and expand their capabilities as needed. For example, the team size can vary drastically without requiring a redesign of the overall system.
- **Limited individual capabilities**: the single robots that compose the swarm usually have simple hardware and software and are only capable to perform simple actions compared to the resulting complex collective behaviour emerging from the numerous interactions among the robots.
- **Robustness and scalability**: robustness in robotics refers to the ability of a robotic system to maintain stable and reliable performance in the face of uncertain conditions.
- **Distributed coordination**: it refers to a robotic system where each component operates independently and collaborates with others to achieve a common goal. Decentralised decision-making is often at the core of distributed coordination.

Thanks to these characteristic and properties, the aim of swarm robotics systems is to be flexible and adaptable to environmental changes, achieving tasks efficiently through decentralisation and self-organised coordination.

The term “swarm” was first introduced in robotics by Beni in 1988 [3], where he laid out the conceptual foundation for a new class of robotic systems composed of autonomous units that cooperate to accomplish tasks. In 1992, Fukuda et al. expanded on this idea by proposing the design of self-organizing robotic systems inspired by biological models [24], coining the concept of cellular robotics. Around the same time, Beni and Wang introduced the term “swarm intelligence” to describe systems in which the collaboration among

individual agents leads to emergent patterns exhibiting complex, non-random behavior.

In the early development of swarm robotics, researchers focused on understanding natural swarming behaviors observed in species such as ants, birds, and fish, with the goal of replicating these patterns in robotic systems [18]. The research drew inspiration from a wide range of biological phenomena, including bird flocking and collective activities of ant colonies.

Numerous studies [28] [18] [20] sought to mimic collective animal behaviours like foraging, flocking, and stigmergy. In this context, foraging refers to the search and collection of resources from the environment, flocking describes the coordinated movement of individuals within large groups, and stigmergy refers to the indirect coordination mechanisms seen in social insects like termites and ants—all being typical animal behaviors. Swarm robotics typically derives engineering principles from natural systems to provide multi-robot systems with similar capabilities [18]

2.3. Blockchain technology

Blockchain technology represents a revolutionary approach to distributed data management and secure information storage, enabling users to conduct economic transactions without depending on a central authority. Essentially, blockchain is a shared, immutable ledger that supports the recording of transactions and the tracking of assets across a network [16].

The first instance of blockchain technology was introduced in 2008 by an individual or group using the pseudonym “Satoshi Nakamoto” with the release of the paper titled “Bitcoin: A Peer-to-Peer Electronic Cash System” [52]. Initially, blockchain was designed as a fully decentralized method to conduct monetary transactions over the Internet via the Bitcoin cryptocurrency. In the following years, however, the emergence of new digital ledgers like Ethereum and various other platforms transformed blockchain into a broader framework for distributed computing. This evolution enabled users not only to record economic transactions but also to execute computer programs, known as smart contracts, on a decentralized computing platform.

Specifically, the Ethereum protocol [30], along with its native cryptocurrency Ether, is among the most widely adopted blockchain platform worldwide that supports smart contracts and the Ethereum Virtual Machine is what defines the rules for computing a new valid state from block to block. Blockchain operates based on the principles of transparency, security, and immutability, making it a distinctive and dependable tool for a

wide range of applications. Transactions are grouped into blocks, which are then linked sequentially to form a chain. Once a transaction is recorded on the blockchain, it becomes extremely difficult to modify, thereby guaranteeing data integrity. The structure of a blockchain is illustrated in Figure 2.2, where three blocks are shown. Each block consists of a header and a body. The header contains the Parent Block Hash, which links it to the previous block, ensuring immutability: if an old block is altered, its hash also changes, making the subsequent chain invalid. The body includes the transaction counter, which specifies the number of transactions (TXs) contained in the block.

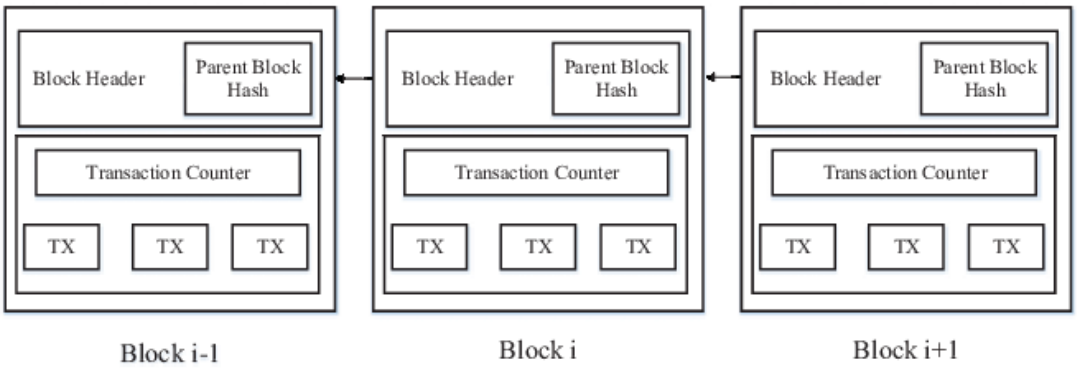

Figure 2.2: Three subsequent blocks of a chain. Figure taken from [79].

Blockchain technology goes beyond cryptocurrencies, being applied in areas such as supply chain management [32], voting systems [29], healthcare records [14], and notably, swarm robotics [19]. Its decentralized architecture removes the need for a central authority while promoting transparency and trust among participants through consensus protocols. Previous studies have demonstrated that decentralized smart contracts executed on blockchain platforms can function as "meta-controllers," enabling secure consensus within peer-to-peer robotic networks [57].

2.3.1. Smart contracts

A smart contract is a set of algorithms—consisting of functions and variables—that is stored and executed on the blockchain. To deploy a smart contract or invoke its functions, users must create and broadcast a transaction within the blockchain network. The blockchain's nodes are responsible for maintaining and updating the internal state of the contract by storing and processing the values of its variables.

Whenever a new transaction is generated, the smart contract is executed by every participant in the network, potentially modifying the values of its state variables. Below is

a brief example adapted from Strobel et al.'s work, "Blockchain technology secures robot swarms" [71]:

Imagine an Ethereum smart contract used to manage the selection of a talent show winner on television. In this case, the audience can vote for their preferred candidate, such as Alice or Bob, by submitting a transaction (for example, including a 0.01 ether contribution) to the show's smart contract on the public Ethereum blockchain. The smart contract records the total votes for each candidate and enforces a programmable condition: once a candidate reaches 100,000 votes, a prize of 1,000 Ether is automatically sent to the Ethereum address associated with that candidate.

This example highlights several benefits of smart contracts over traditional voting processes:

1. The contract conditions and vote counts are entirely transparent.
2. Votes, once cast, are immutable and cannot be altered or removed.
3. The prize is guaranteed to be paid promptly once the specified condition is fulfilled.

2.3.2. The Toychain

Although several blockchain-based smart contract platforms currently exist, in my thesis I employed a simplified blockchain platform called Toychain [74]. Developed in Python at IRIDIA, the Artificial Intelligence Laboratory of the Université Libre de Bruxelles in Belgium, the ToyChain serves as a mock-up blockchain module.

ToyChain abstracts away standard cryptographic mechanisms commonly used in blockchain systems (for example, it does not implement public key cryptography) and focuses only on the components essential for studying how blockchain can be applied to control the behavior of robot swarms. The ToyChain consists of four main components:

- **Node** It represents a user in the network, that has its own blockchain and its mempool. The mempool contains the pending transactions that are not yet in the chain.
- **Block** Transactions are aggregated into blocks, which are then appended to the blockchain. Each block consists of a list of transactions along with the current values of the state variables. Additionally, a block includes a timestamp, a unique cryptographic hash that serves as its identifier, and the hash of its parent block, creating a link between the current block and the previous one in the chain. Figure 2.3 illustrates an example of the information recorded within a block.

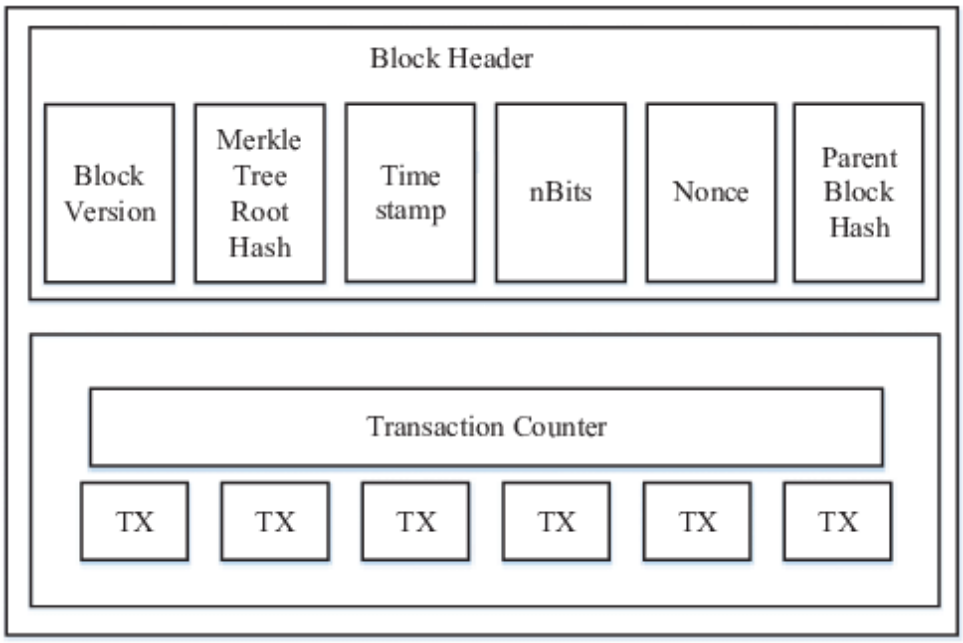

Figure 2.3: Representation the general structure of a single block in a chain. It contains a set of transactions (Tx) and all the other information that are registered into a block [63]. A block in the ToyChain is constructed in the same way [79].

- **TCP and synchronization** The Transmission Control Protocol (TCP) is used in the ToyChain to maintain communications between the robots in the network. At regular intervals and only when communication between two robots is physically possible, the block and the mempool of different chain instances are synchronised.
- **Consensus protocol** The consensus protocol establishes the rules for creating and validating blocks within a blockchain network. All nodes in the system agree to follow these rules, starting from a common point known as the genesis block, which is the initial block identical across all nodes and serves as the foundational starting point of the chain. The specific details of the genesis block depend on the consensus mechanism selected and are determined during the consensus initialization.

  In the ToyChain platform, three consensus protocols are available: proof-of-work (PoW), proof-of-authority (PoA), and proof-of-reputation. In this thesis, I employed the Proof-of-Authority protocol to validate the results. Proof-of-Authority relies on a limited set of trusted nodes, called authorized signers, who produce blocks. The protocol assigns block production rights based on reputation and verified identity, allowing only recognized authorities to participate in block creation and consensus formation.

In essence, Proof-of-Authority ensures that consensus is achieved through cooperation among reputable nodes rather than open competition or resource-intensive computations, offering a secure and efficient mechanism for block validation and network agreement in the swarm robotics blockchain framework.

This aligns with common blockchain consensus principles, where consensus protocols enable all nodes to agree on the ledger state without a central authority, ensuring network security, consistency, and immutability

2.4. State of the art

In this section, I provide an overview of the state of the art in Collaborative SLAM (C-SLAM), describing how C-SLAM frameworks have evolved over time and highlighting recent trends in the research landscape. I also explain why Swarm-SLAM stands out as one of the most advanced and promising frameworks for C-SLAM. Finally, I discuss the growing use of blockchain technology in swarm robotics.

2.4.1. C-SLAM

Robots use SLAM to navigate and map unknown environments. However, since uncharted areas can be vast, the collaboration of multiple robots becomes necessary. When a robot team is deployed, each individual explores a segment of the environment and then shares its findings with the others, collectively building a comprehensive map. C-SLAM offers solutions to several challenges faced by single-robot SLAM systems, such as lowering individual costs, minimizing cumulative global errors, reducing computational demands, and mitigating the risk of single-point failures. Yet, achieving effective coordination among multiple robots remains a complex challenge [9] [35].

These motivations led researchers in the early 2000s to recognize that collaboration between robots simultaneously mapping and localizing themselves presented a promising approach. Alongside advances in multi-sensor fusion technology, existing single-robot algorithms were initially adapted for multi-robot systems, resulting in various successes. Notable achievements include the multi-robot Extended Kalman Filter (EKF) algorithm [62] and cooperative multi-robot localization techniques [53]. Over the following years, the incorporation of multi-source data increased, and the integration of SLAM with deep learning further enhanced system adaptability, reducing failure rates and advancing research in multi-robot collaborative SLAM [9].

The 2016 survey by Saeedi et al. [63] highlighted the emerging predominance of optimization-

based methods over filter-based approaches in multi-robot SLAM, marking a shift in the field's standard techniques. Since then, numerous promising multi-robot SLAM frameworks have been developed, demonstrating significant advances. However, many of these frameworks still face scalability issues, typically performing well only in scenarios involving a limited number of robots. Researchers have therefore focused on addressing critical challenges inherent to collaborative multi-robot SLAM, such as achieving robustness against perceptual aliasing in fully distributed real-time systems [73] and managing the complexities introduced by heterogeneous robot teams. These efforts aim to improve scalability, reliability, and adaptability in more realistic, diverse multi-robot deployments [8].

Today, the term Collaborative-SLAM (C-SLAM) is widely used to describe systems that enable cooperative mapping and localization. Several popular open-source frameworks have been developed under this umbrella, including some of the works previously mentioned: D-SLAM [10], DOOR-SLAM [42], COVINS [65], Kimera-Multi [73], Disco-SLAM [31], LAMP 2.0 [8], maplab 2.0 [12], Swarm-SLAM [40], DVM-SLAM [5] and Ultra-lightweight collaborative SLAM [55].

The above-mentioned architectures can be compared following different fundamental characteristics:

1. **Types of sensors supported:** a SLAM framework can support one or multiple sensors, which characterizes its flexibility. For example, it may employ stereo cameras (s), LiDAR (l), RGB-D cameras (d), or monocular cameras.

2. **Decentralization:** it means that all computation are performed onboard the robot without the need of a central authority.

3. **Robustness:** it refers to the capability of robots to operate reliably under uncertainty and across different environments.

4. **Tolerance to sporadic information:** it's a very important characteristic for these types of systems and refers to their ability to maintain functionality under asynchronous communications and messages that are occurring at irregular intervals.

5. **Sparsity:** it means that robots share only the most relevant data with each other, rather than exchanging all the information they extract, which leads to a more scalable and efficient system.

Recently, Lajoie et al. provided a comparative summary of some existing methods based on these key properties [40]. I present their analysis in Table 2.1, highlighting that fully meeting all these desirable features is quite challenging, and that Swarm-SLAM is the

only framework that accomplishes this.

	Sensor	Decentralised	Robust	Sporadic	Sparse
DSLAM [10]	s	✓			
DOOR-SLAM [42]	s,l	✓	✓	✓	
COVINS [65]	s		✓		✓
KIMERA-MULTI [73]	s	✓	✓	✓	
DISCO-SLAM [31]	l	✓	✓	✓	
LAMP 2.0 [8]	l		✓	✓	
MAPLAB 2.0 [12]	s,d,l		✓		
SWARM-SLAM [40]	s,d,l	✓	✓	✓	✓

Table 2.1: Extensive comparison of C-SLAM open-source frameworks reported in [40].

A recent trend in C-SLAM research is the development of systems that use smaller and cheaper sensors, designed to be mounted on very small devices. For instance, Niculescu et al. created a C-SLAM system using inexpensive and lightweight time-of-flight (TOF) sensors, which measure distances by calculating the travel time of light pulses, mounted on nano-unmanned aerial vehicles (UAV), specifically the Crazyflie 2.1.[55]

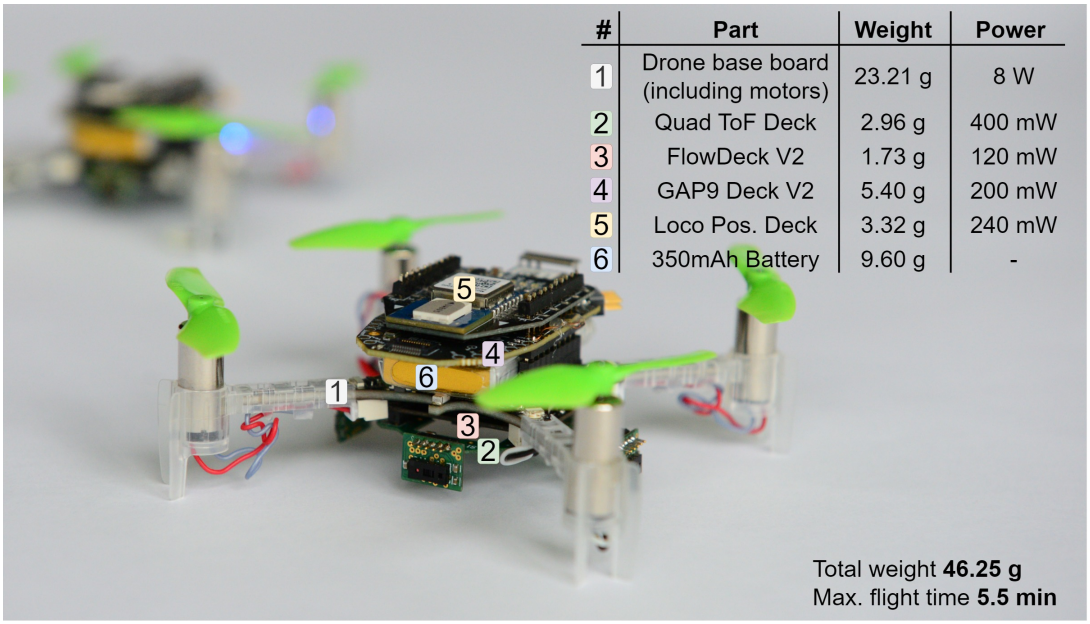

Figure 2.4: The Crazyflie 2.1 UAV equipped with a TOF sensor and other low cost pieces of hardware.

Another example of this trend is DVM-SLAM [5], which uses only Raspberry PI monocular

camera as an onboard sensor for mapping. In this study Bird et al. demonstrated that is possibile to obtain good results in both quality of the generated map and estimated trajectories using inexpensive hardware, ensuring a monetary scalability of the system. Focusing on the hard-constrained and cheap property, Bird et al. also provided a summary of the sensors used in the most famous SLAM systems. I present their analysis in Table 2.2 highlighting that DVM-SLAM is the only decentralized SLAM system using only monocular cameras.

System	Collaboration Type	Monocular	Monocular+IMU	Stereo	Stereo+IMU	LiDAR+IMU	RGBD+IMU
DVM-SLAM	Decentralized	✓					
D^2-SLAM [75]	Decentralized				✓		
Kimera-Multi [73]	Decentralized				✓		✓
Swarm-SLAM [40]	Decentralized				✓	✓	✓
DOOR-SLAM [42]	Decentralized				✓	✓	
D-SLAM [10]	Decentralized			✓			
CCM-SLAM [64]	Centralized	✓					
COVINS [65]	Centralized		✓				

Table 2.2: Comparison of popular C-SLAM open-source sensor configuration.

Although novel frameworks for multi-robot SLAM have been developed recently, Swarm-SLAM [40] is the only framework performing 3D mapping of an unknown enviroment guaranteeing flexibility (it can operate with stereo cameras, RGB-D cameras, and 3D LiDARs), sporadic connectivity (significantly reduced communication demands compared to prior methods), and budgeted strategy for identifying potential inter-robot loop closures through algebraic connectivity maximisation, inspired by recent work on pose graph sparsification [17]. Moreover, the whole architecture is based on the Robot Operating System 2 (ROS2) [47], ensuring compatibility with the majority of robotic systems. They also evaluated the overall system's performance conducting real-world experiments.

Lately, the interest in the study of security in distributed decision-making systems increased significantly [72], and Moroncelli et al. created a Byzantine-fault-tolerant protocol for a C-SLAM system protecting the correct injection of information in the front-end [50]. Previously, however, issues of security and Byzantine fault tolerance had been largely ignored in the C-SLAM literature.

2.4.2. Blockchain secures robot swarms

To address the challenges illustrated by the Byzantine generals problem [44], Strobel et al. [70] were pioneers in addressing scenarios where Byzantine robots—robots exhibiting "Byzantine behaviors" caused by faults or malicious interference—could negatively affect

swarm operations. Detecting such behaviors is challenging, yet they have the potential to disrupt the swarm's ability to reach accurate consensus. Faulty agreements may prevent the swarm from successfully completing tasks, for example, when robots collectively agree on incorrect objectives, leading to wasted resources. Therefore, it is crucial to evaluate the impact of Byzantine behaviors on the swarm's capacity to make reliable collective decisions. In extreme cases, compromised consensus can lead to catastrophic consequences, such as unanimous approval of hazardous actions—an issue highlighted by more recent studies [78].

Secure, decentralized and generalized transaction ledgers [34] appear to offer a promising solution to security challenges in robot swarms [60], as they enable the maintenance of a decentralized database while safeguarding privacy through cryptographic protocols [19]. In [71] and [56], the authors demonstrate how blockchain technology can ensure consensus—the collaborative decision-making process among a group of robots to achieve a common goal or behaviour—even in the presence of Byzantine robots. Drawing inspiration from Bitcoin [52] and Ethereum [30], they developed blockchain-based smart contracts to control the robots and prevent attacks within a swarm. This system enabled the robots to agree on the correct state of the environment that each robot can measure through onboard sensors that are subject to noise. This approach not only enhances security but also shows promising potential for facilitating robot-to-robot transactions within swarms, among various other applications. While most results are customized for specific experimental setups to address particular challenges, there have also been efforts to propose more generalized frameworks that employ blockchain smart contracts to enable robot swarms to securely achieve consensus in arbitrary observation spaces [78]. Detailed experiments and analysis performed on a blockchain-based token economy maintained by 24 physical Pi-puck robots confirm great promises [72].

3 | Swarm-SLAM and its vulnerabilities

In this chapter, I describe the general Swarm-SLAM framework, focusing on its two main modules: the front-end and the back-end. When discussing the front-end, I summarize the studies of a previous master's thesis [49], whereas all the developments on the back-end presented here are original contributions. I then explain their vulnerabilities, highlighting how a Byzantine robot can act to compromise the entire system.

3.1. General framework

The Swarm-SLAM framework is divided into modular sections which are in charge of performing different part of a C-SLAM algorithm and are strictly connected to each others. these modules are described here.

- **Neighbour Management:** The neighbor management module continuously monitors reliable communication neighbors by tracking heartbeat messages—regular signals sent by robots to indicate they are active and reachable. This module works without requiring a fixed number of robots and adapts smoothly to random encounters, providing resilience against disconnections. Additionally, communication and computation budgets are tailored to each robot's capabilities, ensuring efficient use of resources.

- **Front-end:** The front-end module is responsible for processing **odometry** estimates obtained through various methods, along with synchronized sensor data. It extracts both local and global descriptors, supporting a wide range of sensor types. By operating independently of the odometry source, the module ensures flexibility and ease of integration. In Swarm-SLAM, once a scene is registered, the front-end generates a unique global descriptor used for place recognition across robots, automatically associating it with the corresponding odometry keyframe.

  Additionally, the front-end manages the indirect **inter-robot loop closure** detec-

tion process by leveraging communication between robots to identify loop closures in their maps, even when direct observations are unavailable. This process activates whenever robots come within communication range. Although communication and network connectivity can be intermittent, the front-end takes advantage of every opportunity to improve mapping accuracy. When two robots meet and find their descriptors sufficiently similar, one robot computes the geometric transformation (i.e., the loop closure) between the two images linked to those descriptors. This requires exchanging image data; however, to minimize bandwidth usage, only one robot in the encounter receives the other's image to perform the calculations and prevent unnecessary data transfers, as emphasized by the authors of [40].

- **Back-end:** In the back end, intra-robot and inter-robot loop closure measurements, along with odometry data, contribute to the creation of a factor graph called **pose graph** [6]. Local pose graphs are shared with a selected robot for aggregation and optimisation. This decentralised approach ensures that all computation occurs onboard the robots, promoting autonomy and reducing the need for central authority. Optimisation decisions are made through peer-to-peer communication during rendezvous events. The resulting pose estimates are made available periodically as ROS2 messages. In such a system, loop closures are hard constraints in a constrained optimisation problem over a computational graph. Trajectories should be steered towards the ground truth as more constraints are added, thus correcting the poor odometry measurements obtained by the individual robots.

To clarify the general architecture of Swarm-SLAM and illustrate how the different modules interact, I include an image taken from [40] (Figure 3.1).

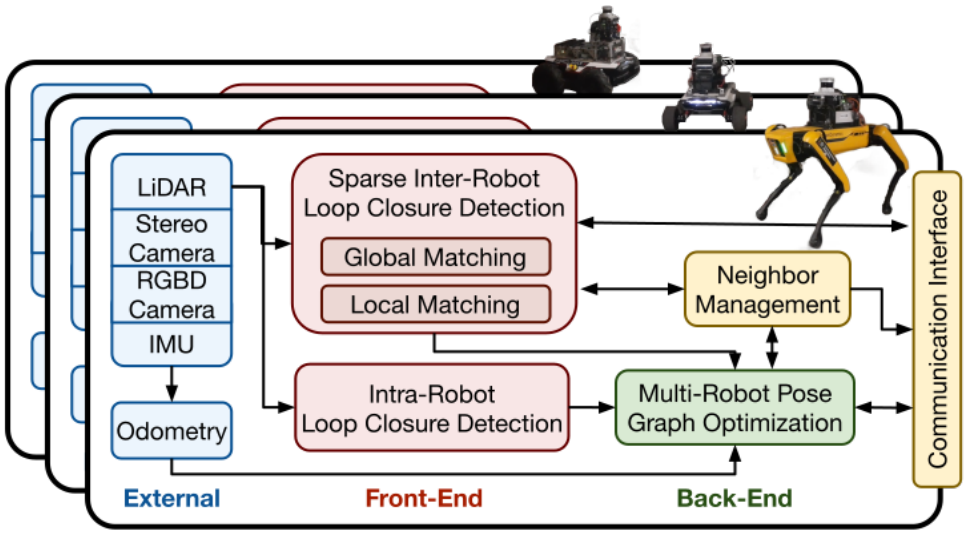

Figure 3.1: General Swarm-SLAM architecture and data flow, every robot runs an instance of the framework including all the modules previously described.

3.2. The front-end

Swarm-SLAM front-end is in charge of producing landmark estimates, odometry measurements, and both intra-robot and inter-robot loop closures [43]. Odometry measurements aim to capture a robot's translation and rotation between consecutive time steps. Common methods include tracking wheel movements, integrating data from an IMU, and performing geometric matching between consecutive images or laser scans. Intra-robot loop closures are measurements used by a single robot SLAM system to relocalize itself and correct the accumulated error caused by odometry drift. Through place recognition, the system identifies previously visited locations and calculates relative measurements between them. Essentially, intra-robot loop closures provide estimates that connect non-consecutive poses along a single robot's trajectory that observed the same place. On the other hand, inter-robot loop closures link poses from trajectories of different robots. They serve as the critical connections that align individual local maps, enabling the construction of a unified global map of the environment. Generating inter-robot loop closures is a primary focus of front-end developments within C-SLAM systems and is essential for maintaining consistency in the collective estimates.

In order to detect inter-robot loop closures, Swarm-SLAM, similarly to other C-SLAM techniques, adopts a two stages approach:

- **Global matching:** for each odometry measurements, the front end saves a connected keyframe. In this way we have a pair odometry-keyframe. For each keyframe,

compact descriptors are extracted from sensor data and can be compared using a similarity score. These descriptors are then broadcast to nearby robots. When two robots encounter each other, they keep track of which global descriptors are already known by the other robot and identify those that still need to be transmitted. Swarm-SLAM uses ScanContext [36] as global descriptors of LiDAR scans and the CNN-based CosPlace [4] for images. It employs nearest neighbour based on cosine similarities for matching.

- **Local matching:** after selecting candidate inter-robot loop closures, the subsequent step is to carry out local matching, also known as geometric verification. This process utilizes a larger set of local features, such as keypoints or point clouds, depending on the sensor type, to calculate the 3D relative pose measurement between the two candidate poses.

In figure 3.2 I summarize global and local matching through a visual example.

Loop closures are a fundamental component of SLAM, as odometry estimates are inherently prone to drift due to sensor imperfections. Relying solely on odometry is insufficient; instead, the pose graph must be continuously refined using loop closures. These act as strong constraints in a later stage of the SLAM pipeline known as Pose Graph Optimization (PGO), significantly improving global consistency. In Figure 3.3 I show different odometry estimates compared to the ground truth, highlighting how the different solutions are different with respect to the ground truth.

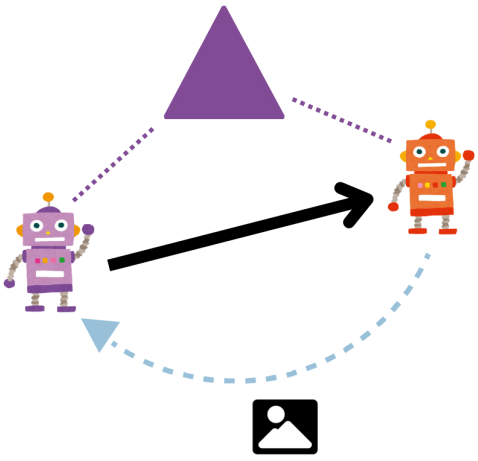

Figure 3.2: Example of an inter-robot loop closure detection: two robots visually recognize a scene in the space (purple triangle), when they meet, they exchange global descriptors until they find a match. If a match is found, then one of the two robots sends a keyframe to the other (light blue dashed line) that is used to calculate the loop closure (black line).

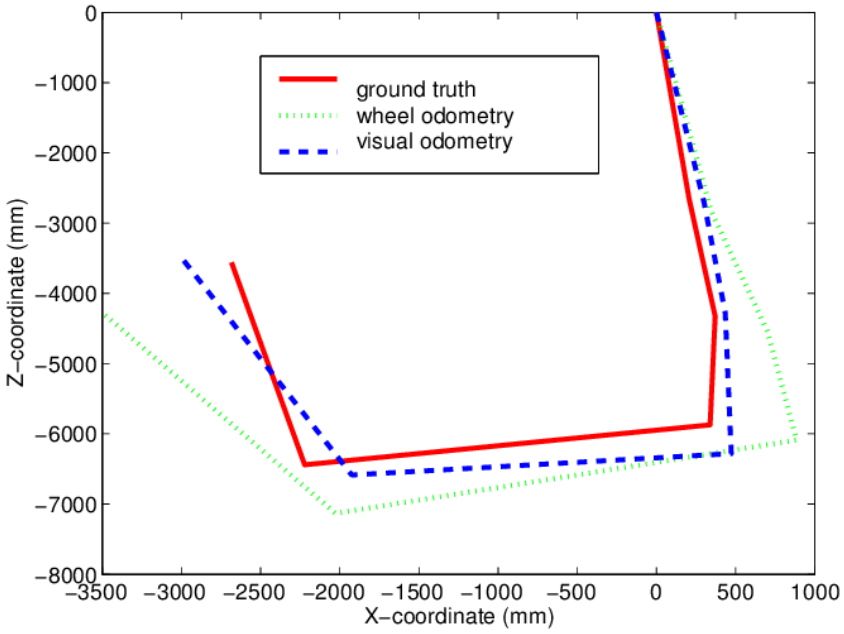

Figure 3.3: Difference between odometry estimates coming from different sources and the ground truth, image taken from [66]. The figure clearly shows that the odometry estimates deviate from the ground truth.

3.3. Security challenges of the front-end

Given the importance of the inter-robot loop closures in a C-SLAM scenario, Moroncelli et al. [50] focused on the ways in which a Byzantine robot can inject false information in the front-end, leading to an inaccurate pose graph. They found two sources of inconsistency:

1. **Odometry-keyframe mismatch:** To determine whether two robots have visited the same location, each odometry measurement must be associated with a corresponding keyframe. During a rendezvous, the robots compare compact descriptors extracted from these keyframes to identify potential place matches. However, if a robot incorrectly associates an odometry measurement with the wrong keyframe, there is a risk that a false loop closure may be established in a future rendezvous. In Figure 3.4 I show an example of this inconsistency.

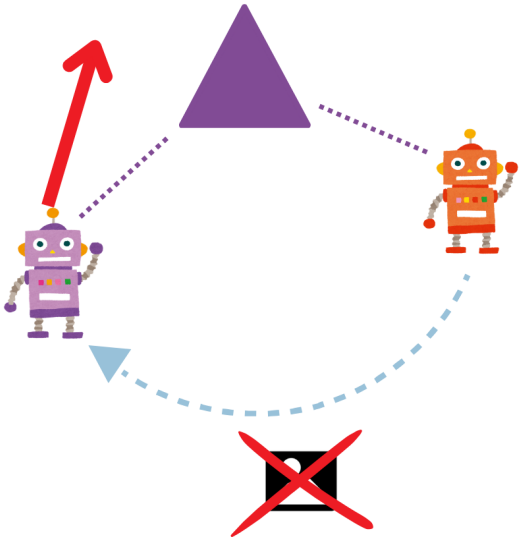

Figure 3.4: Example of odometry-keyframe mismatch, when two robots meet they find a wrong match between their descriptors, consequently one robot sends to the other a wrong keyframe that is used to calculate a wrong loop closure (red arrow).

2. **Incorrect loop closure creation:** in this case we assume that the image sent by one robot is correct, but the other robot calculate a wrong loop closure due to a malfunctioning or a malicious behaviour. I summarize this behaviour in Figure 3.5.

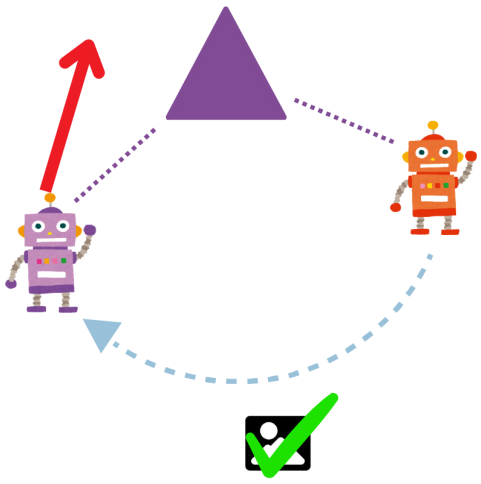

Figure 3.5: Example of incorrect loop closure creation, when two robots meet they find a correct match between their descriptors, consequently one robot sends to the other a correct keyframe but the other robot generates a wrong loop closure due to malfunctioning or malicious behaviour (red arrow).

It is important to address the issue of **perceptual aliasing** [41], which occurs when a robot mistakes two distinct locations as visually similar. This often happens in repetitive environments or when sensor data is limited. Perceptual aliasing can result in false loop closures, where the system wrongly believes the robot has revisited a previously explored place. These incorrect loop closures can significantly distort the map during pose graph optimization, leading to substantial localization errors or even causing the SLAM system to fail.

Although state-of-the-art C-SLAM methods are often described as "robust," this term frequently refers mainly to robustness against outliers caused by perceptual aliasing. Outliers are typically defined as data points that significantly deviate from other measurements. However, when measurements are sparse and differ greatly in magnitude, such outliers are difficult to detect and reject. Solvers based on graduated non-convexity [77], a widely used technique for optimizing non-convex cost functions in pose graph optimization (PGO), are very effective at rejecting spatial perception outliers. Nonetheless, when applied to C-SLAM, these methods remain highly application-specific and mostly address the computer vision problem of place recognition.

Moroncelli et al. [50] investigated the impact of incorrect loop closures on pose graph accuracy using a Gazebo simulation involving eight robots performing random walks in the environment. The robots can recognize eight distinct scenes, which serve as landmarks for computing loop closures. When two or more robots encounter each other, they carry out both global and local matching as previously described to identify loop closures. Subsequently, one robot within communication range is chosen to execute pose graph optimization, leveraging the identified loop closures as hard constraints to continuously refine the map. To model the two previously described sources of inconsistency, they introduced Gaussian noise into the loop closure measurements and compared the resulting pose graph accuracy with the baseline solution where no Byzantine robots were present. As expected the accuracy of the pose graph in term of absolute trajectory error (ATE) is worse in the presence of Byzantine robots compared to the baseline solution. In figure 3.6 I highlight their result, showing how "Byzantine" loop closures can generate a wrong pose graph.

I plotted the results using the EVO software [26] that provides executables and a small library for handling, evaluating and comparing the trajectory output of odometry and SLAM algorithms.

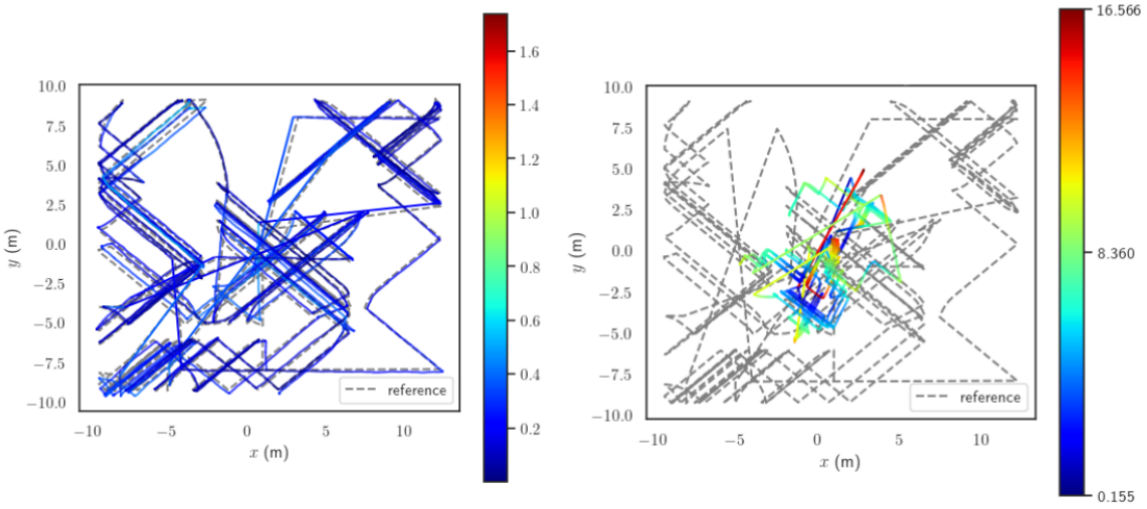

Figure 3.6: Comparison of the absolute trajectory error between the baseline solution and the solution with three Byzantine robots, where Gaussian noise was added to the loop closures. The color indicates the magnitude of the error at each point.

3.4. The back-end

In this section, I describe in detail the functioning of the SLAM back-end, starting from the mathematical problem it solves and the possible approaches to tackle it. I then focus on Swarm-SLAM, explaining how its back-end works and the related security challenges.

3.4.1. Problem statement

C-SLAM back-end is responsible for estimating the robot's state and the map based on measurements provided by the front-end. Unlike single-robot SLAM, it must handle inter-robot measurements, achieve consensus among robots, and often operate without an initial estimate, as the global reference frame and robots' starting positions are typically unknown [43]. In a setup with two robots (α, β), where X_α and X_β are the state variables from robot α and β to be estimated, Z_α and Z_β are the set of measurements gathered by robot α and β independently, $Z_{\alpha\beta}$ is the set of inter-robot measurements linking both robot maps containing relative pose estimates between one pose of robot α and one of robot β in their respective trajectories, and X_α^*, X_β^* are the solutions, the problem can be formulated as:

$$(X_\alpha^*, X_\beta^*) = \arg\max_{X_\alpha, X_\beta} p(X_\alpha, X_\beta \mid Z_\alpha, Z_\beta, Z_{\alpha\beta}) = \arg\max_{X_\alpha, X_\beta} p(Z_\alpha, Z_\beta, Z_{\alpha\beta} \mid X_\alpha, X_\beta) p(X_\alpha, X_\beta) \tag{3.1}$$

However, when the relative starting locations and orientations of the robots cannot be determined, the initial guess of the robots states $p(X_\alpha, X_\beta)$ is not available. In that case, there are infinite possible initial alignments between the multiple robot trajectories. Therefore, in absence of a prior distribution, C-SLAM is reduced to the following Maximum Likelihood Estimation (MLE) problem:

$$(X_\alpha^*, X_\beta^*) = \arg\max_{X_\alpha, X_\beta} p(Z_\alpha, Z_\beta, Z_{\alpha\beta} \mid X_\alpha, X_\beta) \tag{3.2}$$

Similar to single-robot SLAM solvers, C-SLAM back-ends are generally classified into two main categories of inference techniques:

- **Filtering techniques:** filtering approaches are typically described as online methods because they estimate only the current robot pose, marginalizing out all previous poses at each time step. As a result, the posterior estimation at time t depends only on the posterior at time $t-1$ and the new measurements. The most common filtering

technique used for nonlinear problems—such as those encountered in robotics—is the Extended Kalman Filter (EKF). In short, EKFs are Gaussian filters that address the Kalman filter's linearity assumptions by applying linearization, or local linear approximation. However, this linearization can introduce inconsistencies, especially when the noise levels are high.

- **Smoothing techniques:** unlike filtering-based approaches, smoothing techniques improve their accuracy by revisiting past measurements instead of only working from the latest estimate. In smoothing, there is less marginalization required which means that the variables will stay sparsely connected.

Recently, thanks to modern solvers [33] [69], smoothing techniques are preferred in terms of accuracy and efficiency. For a more comprehensive treatment of this topic, the reader is referred to [43].

3.4.2. Swarm-SLAM back-end

In Swarm-SLAM, the back-end adopts a smoothing-based approach, where odometry measurements, as well as intra- and inter-robot loop closures, are incrementally incorporated into a factor graph known as a pose graph. This graph serves as the core data structure for optimizing the entire trajectory history of the robot swarm. A factor graph consists of:

- **Nodes:** Represent the poses of the robot over time.
- **Vertices:** Relation between successive poses

Figure 3.7 shows a simple example of a factor graph, while Figure 3.8 presents a realistic example taken from a SLAM application.

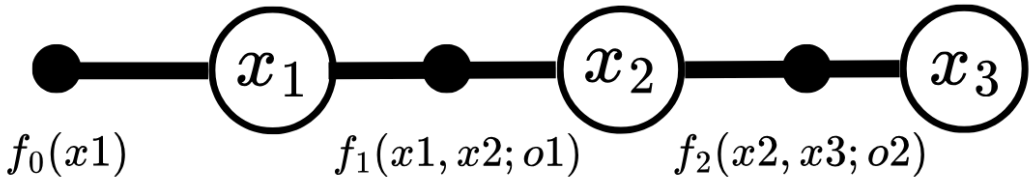

Figure 3.7: A simple factor graph composed by three variables x_1, x_2, and x_3 which represent the poses of the robot over time, rendered in the figure by the open-circle variable nodes. In this example, we have one unary factor $f_0(x_1)$ on the first pose x_1 that encodes our prior knowledge about x_1, and two binary factors that relate successive poses, respectively $f_1(x_1, x_2; o_1)$ and $f_2(x_2, x_3; o_2)$, where o_1 and o_2 represent odometry measurements.

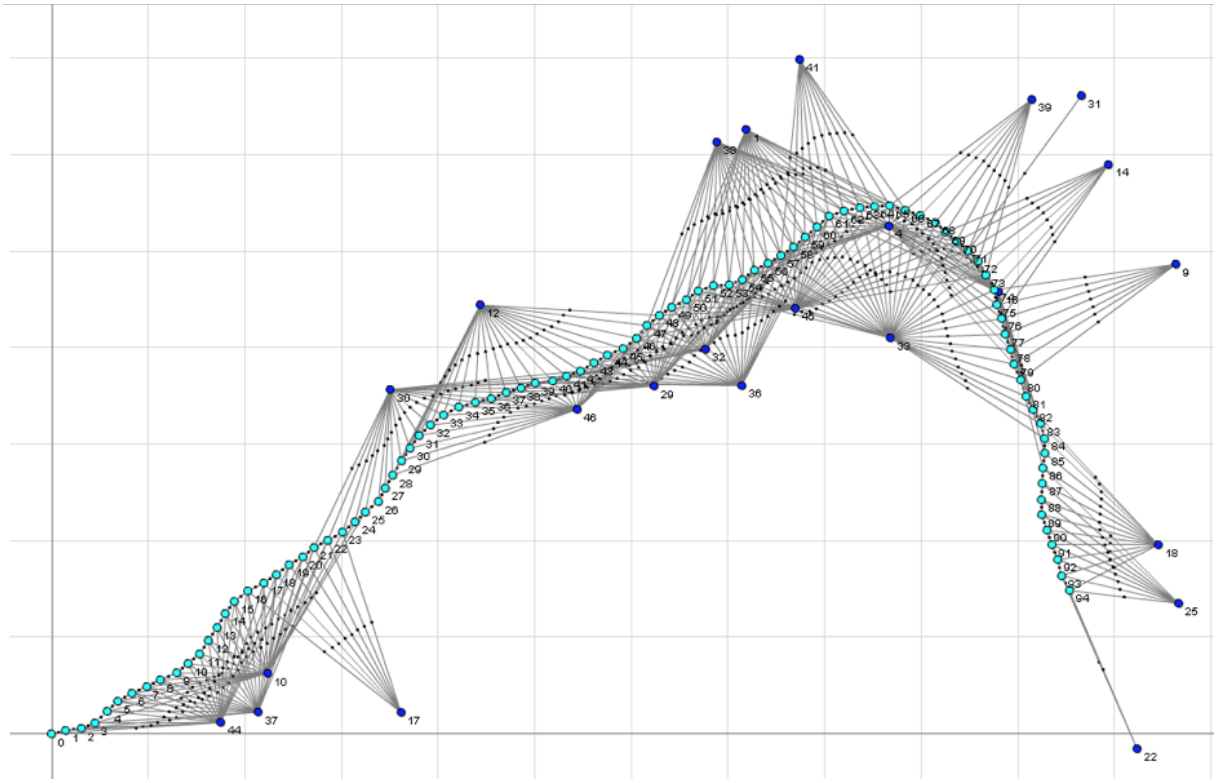

Figure 3.8: Realistic SLAM factor graph. This example is a small excerpt from a real robot experiment in Sydney's Victoria Park [27]. The location of the robot over time is represented by the cyan nodes and the related landmarks are represented by the blue nodes.

In a multi-robot scenario, inter-robot loop closures are vertices connecting poses of different robots, while intra-robot loop closures are vertices connecting two poses which belongs to the same robot. Both of them are used as hard constraints by the pose graph optimizer to refine the map. To better understand how a complete multi-robot factor graph looks like I show an image taken from [22] which shows an example of a two robots factor graphs including odometry and both intra and inter-robot loop closure (Figure 3.9).

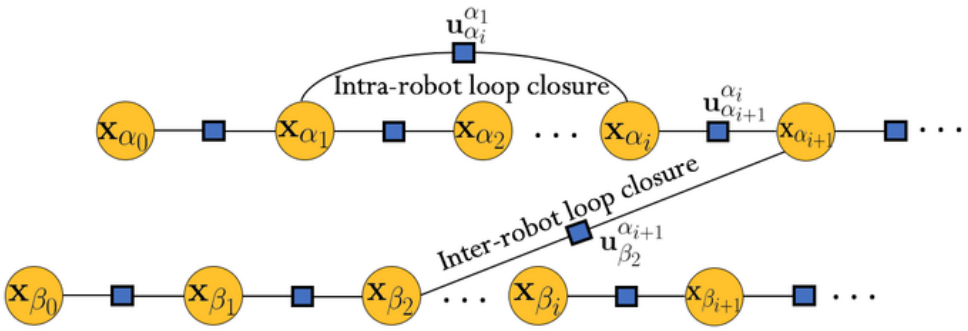

Figure 3.9: Illustration of a multi-robot pose graph for a team of two robots. The yellow nodes correspond to the robot poses, while blue squares represent the relative 3D rigid transformation between the nodes. An inter-robot loop closure is connecting poses belonging to different robots while the intra-robot is connecting two poses of the same robot.

In Swarm-SLAM, each robot is responsible for constructing its own local pose graph, which includes its odometry measurements, and for storing the loop closures it establishes with other robots. The rate at which a single robot adds new vertices in its pose graph via odometry is a configurable parameter, adjustable to match the robot's computational capabilities.

The pose graph optimization (PGO) is performed by a single elected robot. The logic for the election can be customized, Swarm-SLAM by default choose as optimizer the robot with lowest ID (usually robot-0 in the swarm) every time an optimization timer triggers. This timer can also be customized to match the robot's computational capabilites since the PGO is a computationally expensive process and it becomes more and more expensive as long as the pose graph grows in dimensions.

It is important to mention that the PGO works if at least one loop closure has been created between two robots, otherwise we don't have a constraint and the optimization process becomes meaningless. Moreover when a loop closure is created between two robots, those robots become connected and their pose graphs are aggregated and optimized by one of them chosen by the selection rule previously described. This is in order to enhance

decentralization and scalability but I will describe this functioning better later on.

Every time the optimization timer triggers, the PGO starts and it is composed by three phases:

1. **Pose graphs aggregation:** the elected robot receives the pose graphs from the connected robots (they are connected if at least a loop closure has been created between them) and aggregates them in a unique global pose graph.
2. **Pose graph optimization:** after aggregating all the pose graphs from the connected robots, the selected one performs Pose Graph Optimization (PGO) using the GNC optimizer [33], thereby estimating the most likely configuration of the global pose graph.
3. **Optimized trajectory estimates feedback:** after finishing the PGO, the robot optimizer sends back the optimized trajectories to the connected robot. These optimized trajectories are continuously published by the robots in real time via ROS2 topics, making them accessible to users who wish to track them,for instance, for control purposes.

These three phases are summarized in Figure 3.10.

In the beginning, all robots are within their own local reference frames where the origin corresponds to their first pose (i.e., initial position and orientation). Then, when some robots meet for the first time (e.g. robots 0, 4 and 5), we choose the first pose of the robot with the lowest ID (e.g. robot 0) as the anchor. Therefore, as a result of the estimation process, the involved robots estimates share the same reference frame (e.g. robot 0's first pose). In subsequent rendezvous (e.g. robots 2, 3 and 4), the anchor is selected based on the reference frame with the lowest ID (e.g. robot 4's first pose is selected as the anchor since its reference frame is robot 0's). After a few rendezvous, the robots converge to a single global reference frame without requiring rendezvous including all robots (e.g. after the second rendezvous, robots 2 and 3 are also within robot 0's reference frame). This means that Swarm-SLAM can scale to large groups of robots, through iterative estimation among smaller groups of robots.

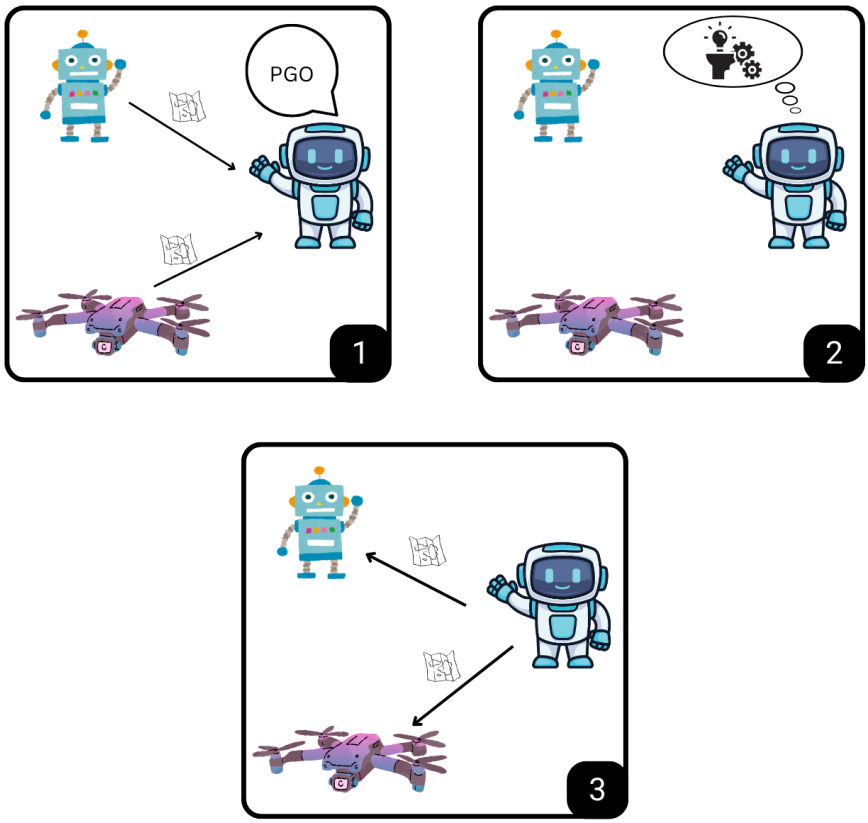

Figure 3.10: PGO phases numbered from one to three. In the first image, the robot in charge of performing PGO requests the local pose graphs from the others in order to aggregate them in a unique global pose graph. In the second image, the robot performs PGO using GNC solver. In the third image, the robot optimizer has finished to perform PGO and gives back the optimized trajectories to the others.

3.4.3. Pose graph structure

The pose graph is stored in a text file following a well-defined structure, where each line encodes specific elements such as nodes (poses) or edges (constraints). Edges come from odometry estimates that link two consecutive poses of the robot and from intra and inter-robot loop closures which connect two non-consecutive poses of the robot's trajectory. Two common structure that can be found in various SLAM application are:

1. **KITTI:** in KITTI format, every line corresponds to one robot's pose and it is represented by a 3x4 tranformation matrix
2. **TUM:** also in TUM every line corresponds to a robot's pose, but differently from

KITTI, it is represented with a timestamp, three values for translations and four values for rotations (quaternion).

Swarm-SLAM uses an extension of the TUM format called g2o [38] which extends the TUM format with edges that encode odometry estimates and loop closures. With `VERTEX_SE3:QUAT` we can define the nodes of the pose graph and with `EDGE_SE3:QUAT` we can define the edges connecting different poses. We have no flexibility on the pose graph format since the GNC optimizer used by Swarm-SLAM requires only g2o files as entry.

Table 3.1: Example of `VERTEX_SE3:QUAT` and `EDGE_SE3:QUAT` entries in a `.g2o` pose graph file taken from a real application of Swarm-SLAM.

Field	**Description**	**Example Value**
VERTEX_SE3:QUAT		
Tag	Entry type	VERTEX_SE3:QUAT
ID	Node identifier	18858823439615201
x	Translation X [m]	195.491
y	Translation Y [m]	-33.4011
z	Translation Z [m]	87.7578
q_x	Quaternion X	-0.294046
q_y	Quaternion Y	0.132671
q_z	Quaternion Z	0.877308
q_w	Quaternion W	-0.355338
EDGE_SE3:QUAT		
Tag	Entry type	EDGE_SE3:QUAT
ID_from	Source vertex ID	18858823439615201
ID_to	Target vertex ID	18858823439615202
Δx	Relative translation X [m]	-0.0527279
Δy	Relative translation Y [m]	0.675142
Δz	Relative translation Z [m]	-0.00520583
Δq_x	Relative quaternion X	0.000833464
Δq_y	Relative quaternion Y	-3.65699e-05
Δq_z	Relative quaternion Z	0.00753978
Δq_w	Relative quaternion W	0.999971
Information(1,1)	Information matrix term	100
Information(1,2)	$\cdots$	0
$\vdots$	Remaining information terms	$\cdots$

3.5. Security challenges of the back-end

As previously mentioned, pose graph optimization is carried out by the robot with the lowest ID among those connected through loop closures. Moroncelli et al. [50] assumed global communication among the robots to simplify the integration of blockchain technology into Swarm-SLAM. Although this assumption facilitates the integration of blockchain into Swarm-SLAM, it also introduces a single point of failure, In fact, if the optimizer robot is Byzantine, it can generate a poorly optimized map, leading all other robots to rely on incorrect trajectory estimates, potentially resulting in catastrophic consequences.

One contribution of my research is to characterise the different ways in which a Byzantine robot could harm the pose graph generation and optimization process. I identify several potential sources of inconsistency, ranging from the input data provided to the PGO, to the optimization process itself, and the resulting output. Figure 3.11 illustrates a schematic representation of these inconsistencies.

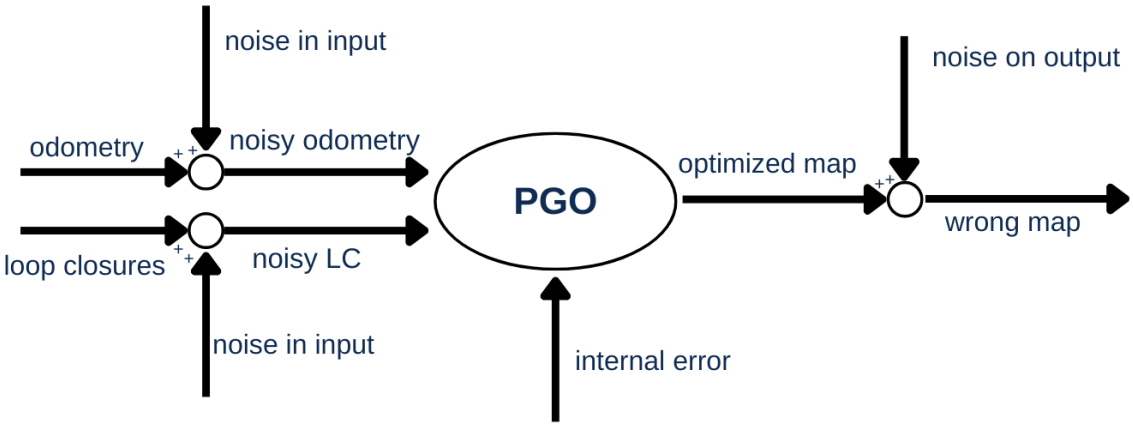

Figure 3.11: Scheme of the various sources of inconsistency that could act on the pose graph optimization.

- **Incorrect input data:** Pose graph optimization is a constrained optimization process that operates on a factor graph composed of odometry and loop closure edges. A Byzantine robot may inject noise into this input data, potentially resulting in a poorly optimized map. In previous sections, we discussed a Byzantine fault-tolerant protocol that safeguards the system against malicious loop closures. This study builds upon that work by extending the protection mechanism to also address the issue of Byzantine odometry estimates.

- **Internal parameter corruption:** By internal error, I refer to some form of tampering or unintended modification of the internal parameters of the PGO process. A detailed explanation of these internal parameters will be provided later in this thesis.
- **Incorrect output transmission:** This refers to a type of tampering or disturbance that can occur in the well-optimized pose graph, causing distortions. The pose graph is structured so that edges connect different robot poses. A Byzantine robot could behave honestly during the PGO process but later act maliciously by modifying the edges of the graph, thereby corrupting it.

In the following subsections, I provide a detailed explanation of each source of inconsistency, in order to characterize the specific form of Byzantine behavior associated with each case. To explain the details, I first illustrate how a pose graph is encoded and its structure.

3.5.1. Incorrect input data

By "incorrect input data", I refer to the intentional injection of artificial noise into the PGO inputs (odometry and loop closures) by a Byzantine robot. We have already seen the problem of noise injection in the loop closures in Section 3.3; here, I focus on the drift that can be added to the odometry estimates. From a real world perspective this behaviour models faulty sensors that could provide poor readings or a malicious robots that wants to distort the global pose graph. I simulated this source of inconsistency with ARGoS simulator [58] employing five robots making a random walk in a unknown environment, four of them with a very accurate odometry estimation and one Byzantine robot with a noisy odometry. Figure 3.12 shows the outcome of this injection of noise in the odometry for one robot. I created the loop closures as soon as two robots are in communication range.

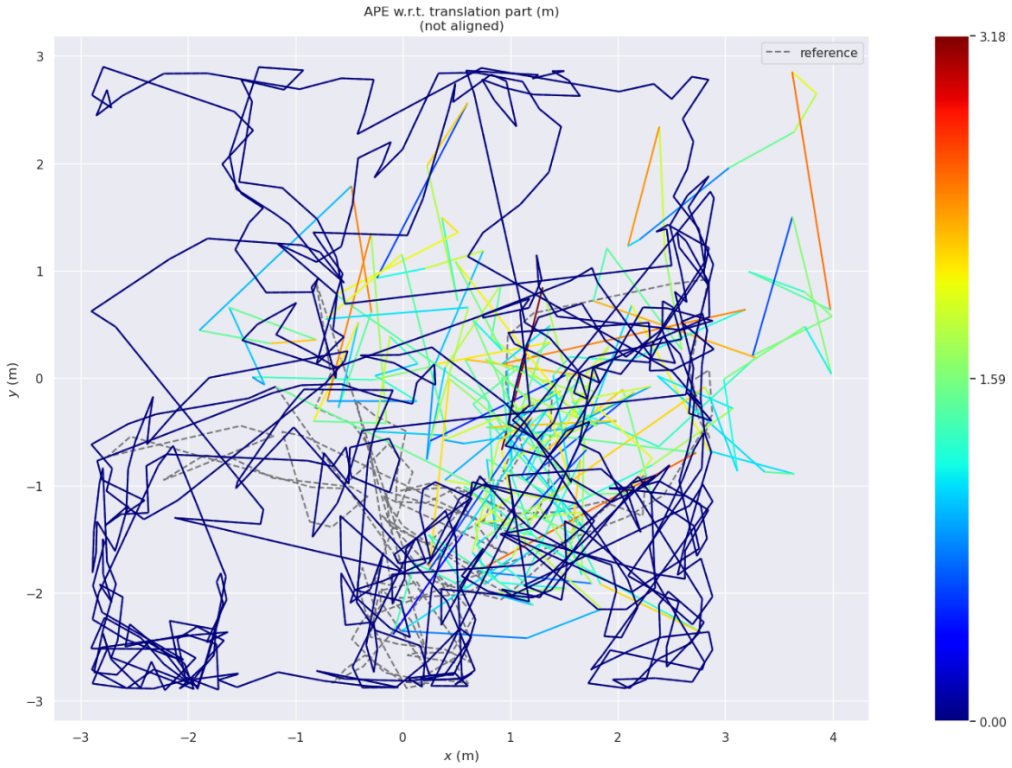

Figure 3.12: Trajectories of five different robots where four of them are trustworthy and one of them is Byzantine. The image shows that the trajectories of the four honest robots are identical to the ground truth, in fact, the dashed grey line representing the ground truth and the blue line representing the estimated trajectories of the honest robots are overlapping, while the Byzantine one is adding noise to its estimates leading to a big absolute trajectory error as we can see from the colours that shows the magnitude of error for each point.

By running the GNC optimizer [33]—a robust optimization algorithm introduced in the GTSAM library [15] to address non-linear optimization problems in the presence of outliers—on the pose graph shown in Figure 3.12, I observed that it fails to adequately refine the graph. This indicates that even advanced solvers like GNC struggle to handle large sources of noise in the odometry measurements. In Figure 3.13 I show the resulting pose graph after optimization.

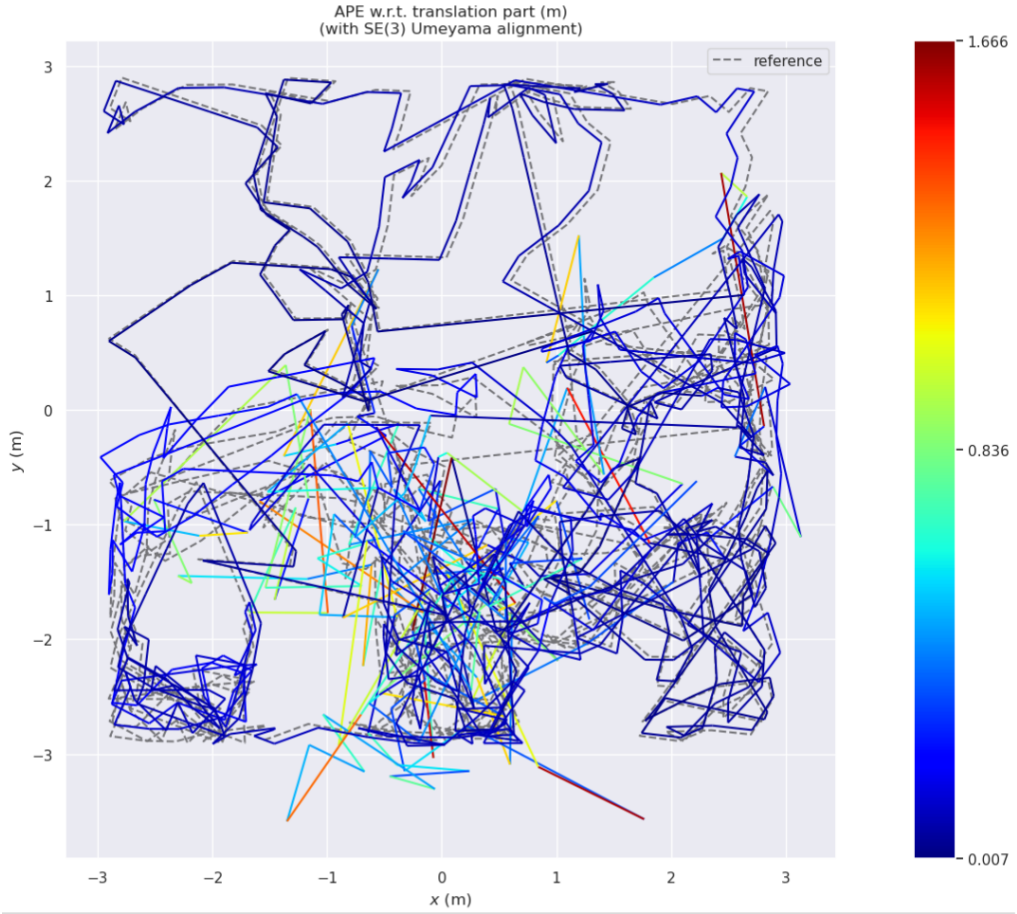

Figure 3.13: The resulting pose graph after optimization, shown in the figure, demonstrates that the optimizer fails to efficiently refine the map. In fact, several points still present high errors, as indicated by the colors that represent the error magnitude at each node. Moreover, the optimization slightly perturbs trajectories that were previously well aligned with the ground truth.

3.5.2. Internal parameter corruption

As previously mentioned, by internal parameter corruption I refer to some form of tampering or unintended modification of the internal parameters of the PGO process. My study involves identifying parameters that significantly influence the optimization result. A Byzantine robot could tamper with these parameters, leading to an inaccurate optimized pose graph. An important parameter is the **covariance matrix**. The covariance matrix essentially gives a weight to each odometry edge and loop closure during the PGO given our trust on them. If the covariance associated with an edge is high, it indicates low confidence in that measurement, and consequently, it should have a low weight during the PGO, viceversa, if the covariance is low, it means that we can give high trust on the measurement and it should weight more during the pose graph optimization. The covariance is a value that we can choose based on the calibration and testing of the sensors, if during calibration, it is observed that one sensor is less accurate than another, the covariance matrix values associated with the less accurate sensor should be set higher than those of the more accurate one.

In order to study the impact of the covariance into the system, I run a series of experiments assigning to each robot different levels of covariance. I performed these tests using GRACO dataset [80], a dataset collected to help in the SLAM system's development and evaluation. It consists of a real world dataset collected by a group of ground and aerial robots equipped with Light Detection and Ranging (LiDAR), cameras, and Global Navigation Satellite/Inertial Navigation Systems (GNSS/INS). All sensors were well-calibrated and triggered by a self-made synchronization module, and the synchronization accuracy can reach millisecond level.

During testing, I observed that modifying the covariance values from the default settings used in Swarm-SLAM PGO—either increasing or decreasing them—can result in a faulty map that significantly deviates from the ground truth. This implies that a Byzantine robot optimizer could manipulate these covariance values, potentially leading to catastrophic outcomes. In Figure 3.14, I present one of the several results obtained from these tests.

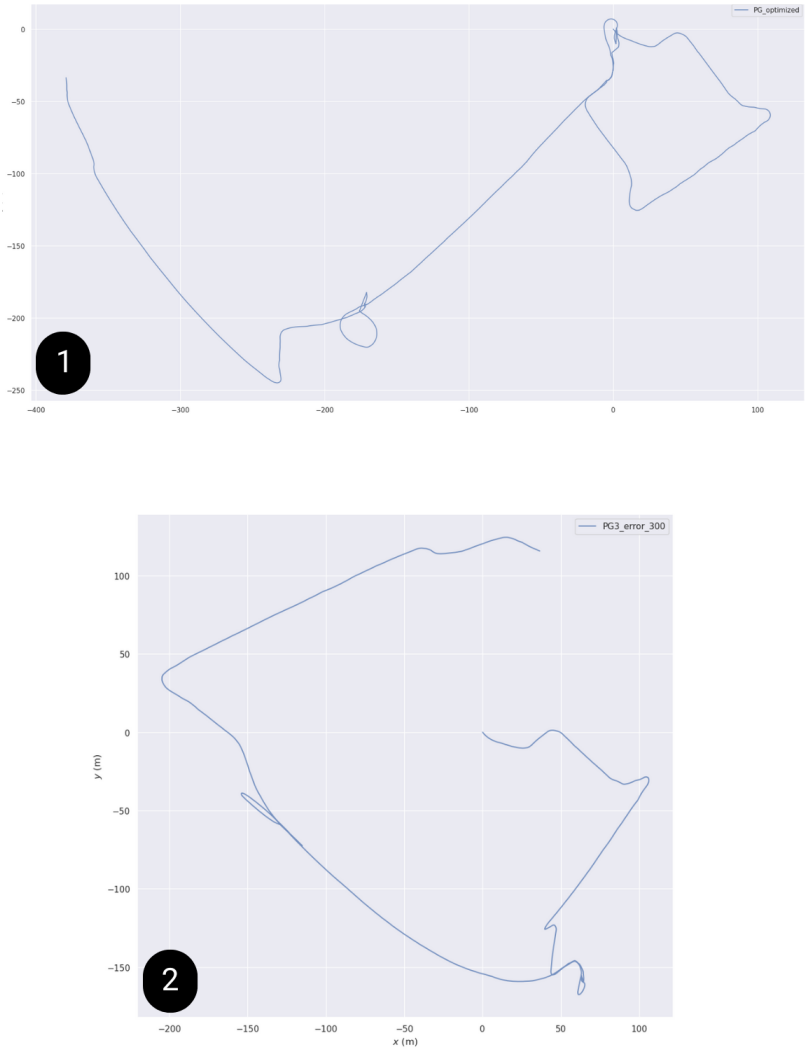

Figure 3.14: Results of a test with altered covariance values for odometry estimates. The first image shows the optimized pose graph using the default covariance value used in Swarm-SLAM (100), which corresponds to a relatively high trust in the odometry measurements; the result is almost identical to the ground truth. The second image shows the optimized pose graph after increasing the covariance to 500, which reduces the trust placed in the odometry data. As a consequence, the optimization relies less on odometry and deviates more from the ground truth, leading to a clearly distorted trajectory

3.5.3. Incorrect output transmission

This is the final type of potential inconsistency examined in this study that may occur in the presence of Byzantine robots. In this case, I assume that all robots operate correctly in the front-end; that is, no robot produces significantly inaccurate odometry estimates, and all loop closures are valid. I also assume that no internal errors are present, meaning there is no tampering with the covariance values. Under these assumptions, the back-end is expected to function correctly, and the optimized map should be continuously refined. However, a Byzantine robot optimizer could wait for the correct outcome of the PGO and subsequently inject artificial noise into the poses and edges of the factor graph. It is worth noting that the pose graph is stored as a text file, making it relatively easy for a malicious robot to access and modify it. In Figure 3.15 I show that this inconsistency can lead to a completely distorted pose graph.

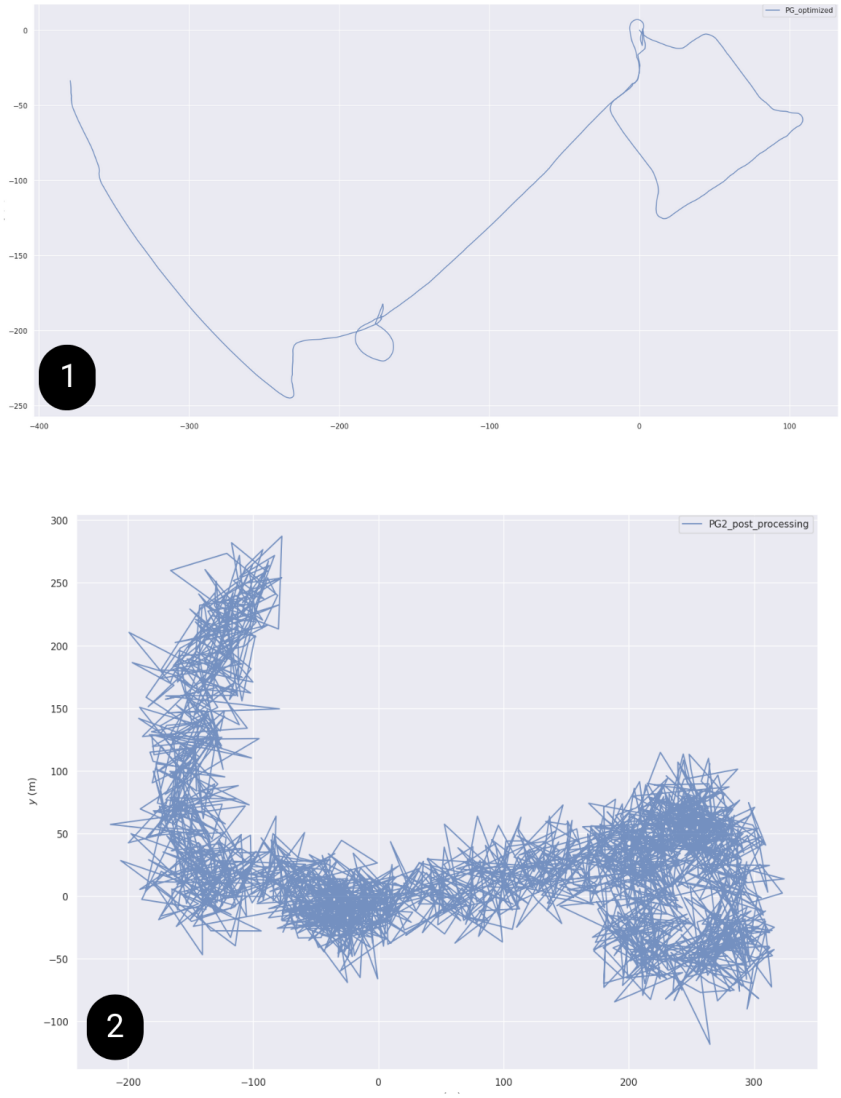

Figure 3.15: Difference between the correctly optimized pose graph (panel 1) and the same pose graph after being tampered with by adding Gaussian noise to its poses (panel 2). The GRACO dataset was used to run Swarm-SLAM experiments.

4 | Protecting Swarm-SLAM

In this chapter, I present the proposed solutions to the problems outlined in the previous chapter. I begin by summarizing the key aspects of an existing Byzantine fault-tolerant protocol designed to ensure the correct injection of loop closures into the system. I then present my original contribution, which focuses on enhancing the security of the back-end.

4.1. Protecting Swarm-SLAM front-end

The following discussion focuses on protecting the Swarm-SLAM front-end, with particular attention to the protocol that safeguards loop closure injection against Byzantine robots. I also present the positive results achieved by employing this protocol in a Gazebo simulation.

4.1.1. Byzantine fault-tolerant protocol

Recognizing the critical role of loop closures in constructing a consistent pose graph, Moroncelli et al. [50] developed a Byzantine fault-tolerant protocol to verify the validity of loop closures before injecting them into the system. In a Byzantine fault-tolerant system, the design principles ensure resilience against erroneous and potentially malicious behaviors from a certain proportion of nodes or components within the system, including those that may intentionally transmit incorrect information.

The building blocks of a Byzantine fault-tolerant system are [44]:

- **Replication:** multiple copies or versions of critical information should exist to fight against malicious attacks.
- **Voting or consensus mechanism:** the entities of the distributed system must reach an agreement to maintain effective decision-making capabilities.
- **Majority agreement:** decisions are taken based on democratic principles.
- **Redundancy and diversity:** redundancy should be present at every level, ensuring that the system can continue functioning even if some components fail.

- **Cryptographic security:** privacy and authenticity must be preserved.

To ensure system decentralization and meet these requirements, blockchain technology is employed. Blockchain technology, introduced in 2.3 generates and maintains an immutable digital ledger that records transactions among agents in a peer-to-peer network [19]. While initially designed for financial transactions between humans, blockchain has been applied across a wide range of domains. In contrast to a centralized controller, blockchain offers several benefits: its decentralized nature allows easy integration within Swarm-SLAM by distributing control of information throughout the swarm, thereby eliminating the risks associated with relying on a single central server, a scenario unsuitable for Swarm-SLAM. Additionally, blockchain ensures that stored data is tamper-resistant, preventing unauthorized modifications by malicious robots. Finally, with blockchain technology we have access to smart contracts, programs that reside on the blockchain and can be executed by all the participants.

In this implementation, ToyChain's smart contracts are used to filter out false information originating from the robots. Each robot acts as a ToyChain node, collectively maintaining the blockchain network through simulated peer-to-peer communications. The consensus protocol employed is Proof of Authority (PoA), the standard protocol used by Toychain, where robots authorized to validate transactions are selected based on their reputation within the network. In contrast, Proof of Work (PoW)—the protocol used by Bitcoin —relies on computational power, which is often impractical in robot swarms due to their limited computational capabilities. Since Swarm-SLAM already demands significant computational resources for data processing and optimization, choosing PoA is the most sensible approach.

When Swarm-SLAM is secured by the ToyChain, loop closures need to pass a filtering mechanism before being processed by the back-end. Every robot that wants to propose a loop closure need to make a transaction to the Toychain. Once consensus is reached and the smart contract execution results in the approval of a new set of loop closures, these loop closures are accepted by the robot responsible for optimizing the pose graph. Figure 4.1 depicts the workflow of the ToyChain when a transaction is registered from a robot.

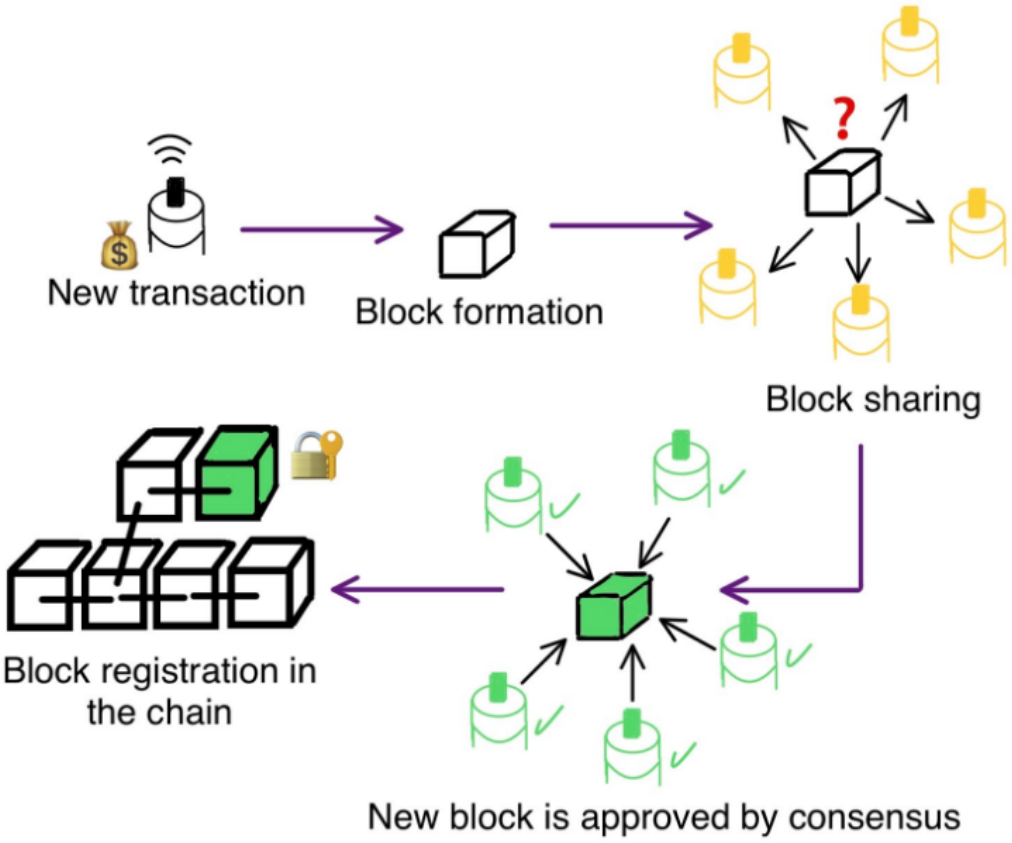

Figure 4.1: Workflow of the ToyChain when a transaction is registered from a robot. In step 'New transaction' it is represented a robot that wants to propose a new transaction, that in this case corresponds to the registration of an inter-robot loop closure. The robot must stake some amount of digital asset to register the transaction (the robot posses a total amount of wealth represented by the money bag). Following the sequence dictated by the purple arrows, the step 'Block formation' represents the block production, where multiple transactions are grouped together and registered in a block and added to the blockchain. At each regular interval, each node requests information about the blockchain and the mempool of its peers. So, blocks synchronisation happens and the information is shared across the network at step 'Block sharing'. Before the block registration into the chain, the new block must be approved by a consensus protocol. I use Proof of Authority in step 'New block is approved by consensus'. In step 'Block registration in the chain', a block is registered and signed into the blockchain in a tamper-proof and immutable manner. Figure taken from [49].

A transaction registers and makes public sensible data that compose the loop closure:

- **Descriptor of the scene:** Which scenes in the map are recognized by the robot;

the map contains a total of nine scenes that the robot can recognize.

- **ID sender:** which robot is sending the keyframe to perform local matching.
- **ID receiver:** which robot receives the keyframe.
- **Pose of the sender:** pose of the robot (x, y position) sending the keyframe at the moment it recognized the scene.
- **Pose of the receiver:** pose of the robot (x, y position) receiving the keyframe at the moment it recognized the scene.
- **keyframe sent:** keyframe sent by the robot to perform local matching.
- **Transformation:** geometric transformation from the pose of the robot receiving the keyframe to that of the robot sending it. During the entire process, an inter-robot loop closure is treated as high-level and lightweight information that represents a geometric transformation between two robot poses, based on raw and heavy data, e.g., images or LiDAR point clouds.

When the ToyChain receives a transaction, it triggers a smart contract that performs a consistency check on the loop closures using geometric verification. A loop closure represents a geometric transformation from one robot's pose to another's, modeled as an arrow. If there are at least three loop closures associated with the same scene in the map and involving three different robots, the smart contract verifies whether these loop closures form a triangle. If a triangle is formed, the loop closures are validated and allowed to enter the pose graph optimization; otherwise, they are rejected and excluded from the optimization process. Furthermore, each robot is required to spend some cryptocurrency to initiate a transaction. If a triangle of loop closures is successfully validated, the involved robots are refunded their cryptocurrency; otherwise, they lose it. This mechanism serves as a reputation system, effectively excluding Byzantine robots from initiating new transactions. Malicious behavior causes their cryptocurrency balance to continuously decrease until they eventually lack sufficient funds to perform further transactions. In Figure 4.2 I make a visual example of this filtering method.

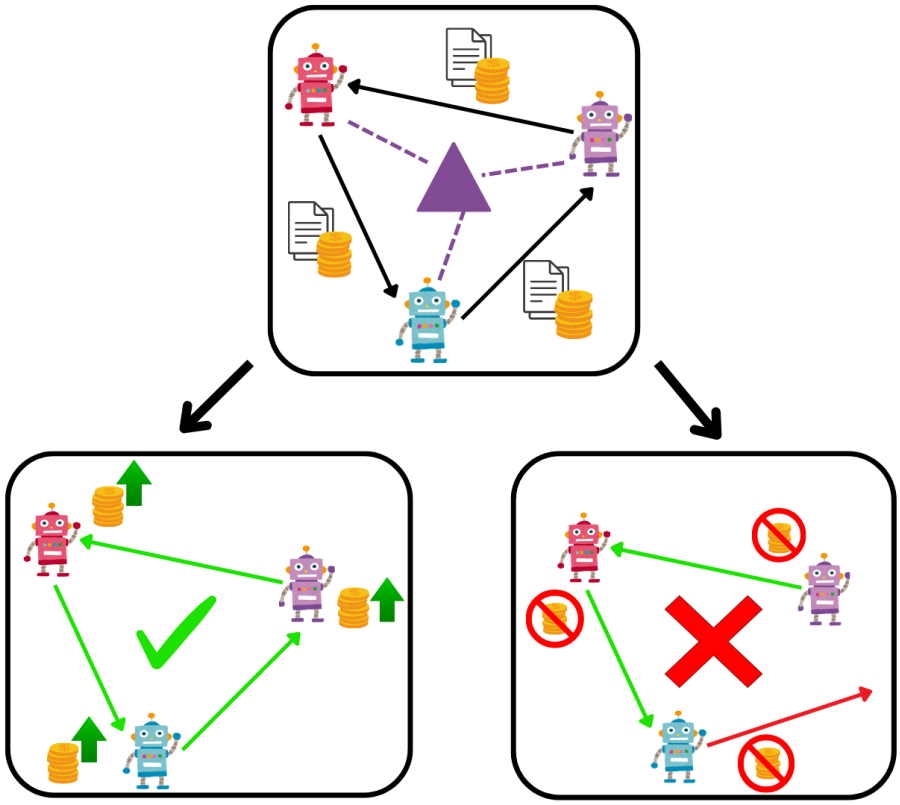

Figure 4.2: Visual example of geometric verification where each robot spends some cryptocurrency to initiate a Toychain transaction. Upon receiving data, a smart contract is triggered to verify whether the loop closures form a triangle. If such a triangle is formed, the loop closures are validated and the robots are refunded their cryptocurrency. However, if a robot maliciously proposes incorrect loop closures causing the triangle not to form, the loop closures are rejected, and the robots do not get their cryptocurrency back. This process ensures the reliability of the loop closures through geometric consistency and discourages malicious behavior.

Thanks to this filtering mechanism, only verified loop closures can enter the pose graph optimization and through the reputation mechanism, Byzantine robots cannot inject false information as soon as they run out of cryptocurrency. An important feature of this Byzantine-fault-tolerant protocol is the **security level parameter**. The security level indicates how many times an inter-robot loop closure has been validated by a different triplet of robots and hence, how secure it is. When a new loop closure is proposed, its security level is zero. When the loop closure becomes part of a validation triangle with other two loop closures proposed by two different robots, its security level is set to one.

Each time the loop closure is included in other validation triangles involving different robot pairs, its security level increases by one. In our work, as soon as a loop closure reaches the security level one, it is included in the PGO and the smart contract returns the deposited authorisation tokens to the robot. This means that we secured the system from individual Byzantine robots, but not from colluding ones. However, the minimum security level can be increased to protect the system against potential collusion of Byzantine robots, albeit at the cost of a higher latency. The smart contract also gives a reputation token to the robots each time their loop closures lead to a security level increment. Therefore, robots that submit valid loop closures increase their reputation over time as more peers validate the loop closures, increasing their security level. In Figure 4.3 I make a visual example to better explain this feature.

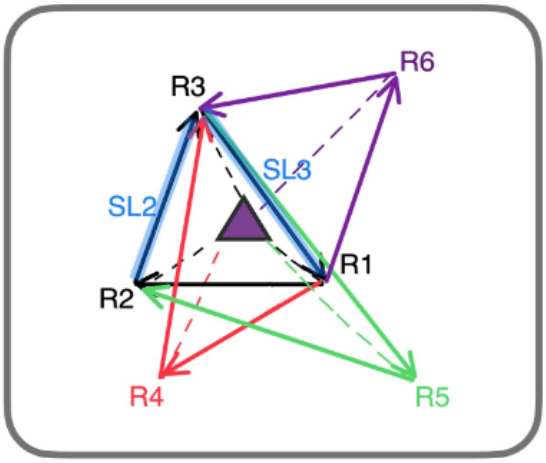

Figure 4.3: Visual representation of the evolving process as multiple triangles are created. The security level (SL) example in the text is extended: six robots (R1, R2, R3, R4, R5, and R6) recognised the same scene (the purple icon), and four triangles (black, red, green, and purple) are measured and verified. In particular, the black is the first triangle validated, its loop closures were created early in time with respect to the others. Subsequently, as soon as new loop closures are proposed by different robots, they complete new triangles with the loop closure already present. For instance, two green loop closures appear and they complete a green triangle with the black loop closure. All loop closures have at least security level equal to 1 because they all participate in one triangle validation. Loop closures with SL bigger than 1 are highlighted in light blue. The loop closure from R2 to R3 has SL of 2 because it is involved in two triangles (the black and the green). The loop closure from R3 to R1 has SL of 3 because it is validated three times from the black, green and purple triangles. Notice that, for example, security level equal to 3 is possible when three triangles are approved from different sets of robots. Hence, the number of agents required to reach SL of 3 is five. Image taken from [49].

4.1.2. Simulation results

This Byzantine fault-tolerant protocol was tested in a Gazebo simulation [37] employing eight TurtleBot3 Waffle model robots [1] and with eight scene (that works as landamarks) that the robots can regognize in order to calculate inter-robot loop closures with the others. Each robot operates with its own noisy odometry data, runs an instance of Swarm-SLAM, and manages blockchain functionalities. During rendezvous, the robots exchange information, compute loop closures, and synchronize their blockchain states. While honest robots generate accurate loop closures, Byzantine robots introduce noise into the geometric transformations. Figure 4.4 shows the Gazebo map, where the white lines represent the walls, the red triangles indicate the nine scenes present in the map, and the blue dots represent the robots.

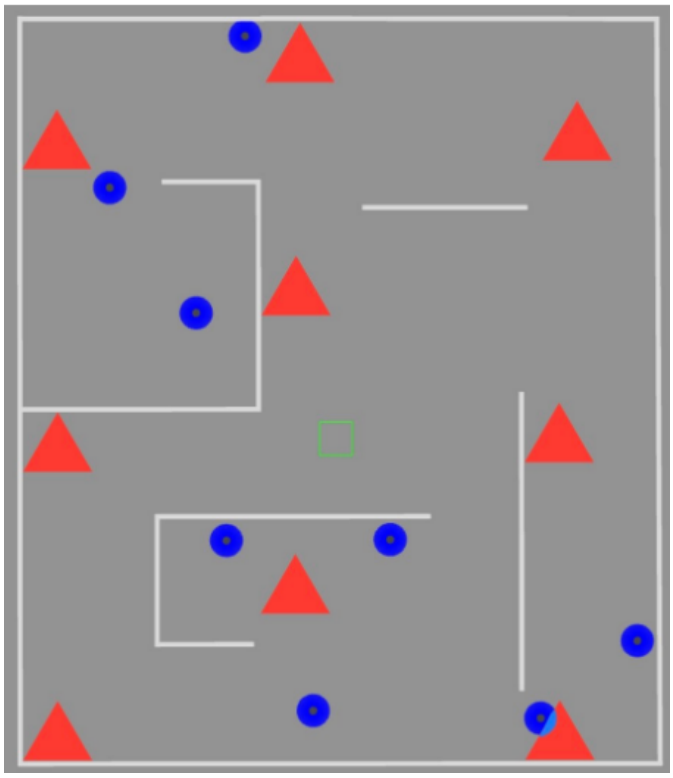

Figure 4.4: Gazebo simulation with eight TurtleBot3 Waffle robots (blue dots) and eight scenes (red triangles) used as landmarks for computing inter-robot loop closures.

Initially, tests were conducted without Byzantine robots to verify whether the loop closures could compensate for the intrinsic noise in the odometry. The results, shown in Figure 4.5, demonstrate how loop closures significantly enhance the accuracy of the pose graph.

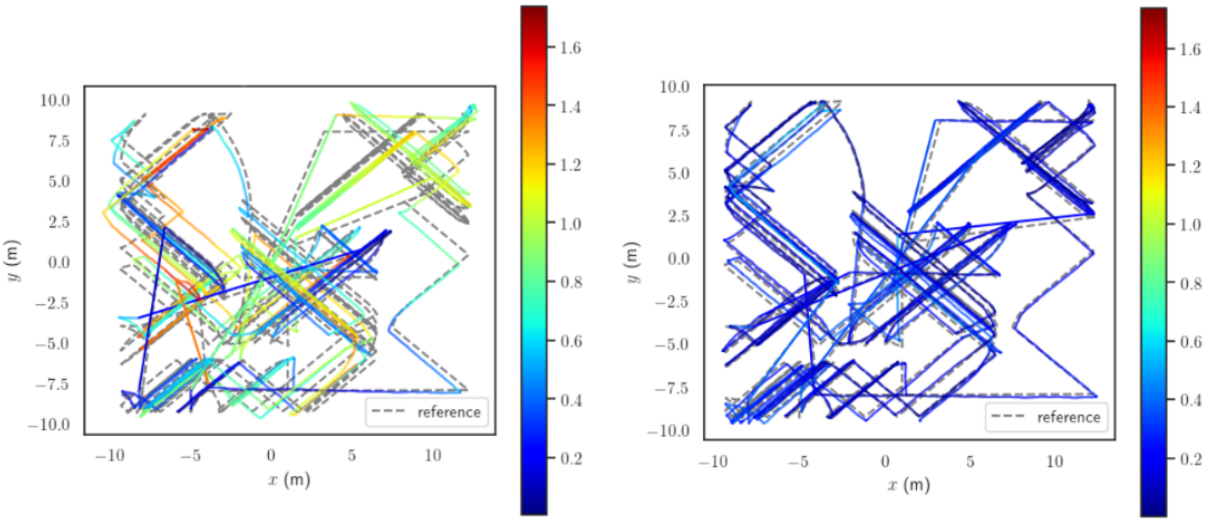

Figure 4.5: Comparison between the non-optimized pose graph (left), where odometry noise is impacting on the absolute trajectory error, and the optimized solution (right) where loop closures succesfully enhance the quality of the pose graph. the grey dashed line is the ground truth and the color indicates the magnitude of error for each point.

Subsequently, tests were conducted including a variable number of Byzantine robots and the protection mechanism explained in the previous section. In Figure 4.6 I highlight the good results obtained. In fact even in the presence of five Byzantine robots (out of a total of eight), the security mechanism is rejecting "Byzantine" loop closures and this leads to an accurate pose graph at the cost of higher latency (because loop closures need to pass through the filtering mechanism before being added to the pose graph).

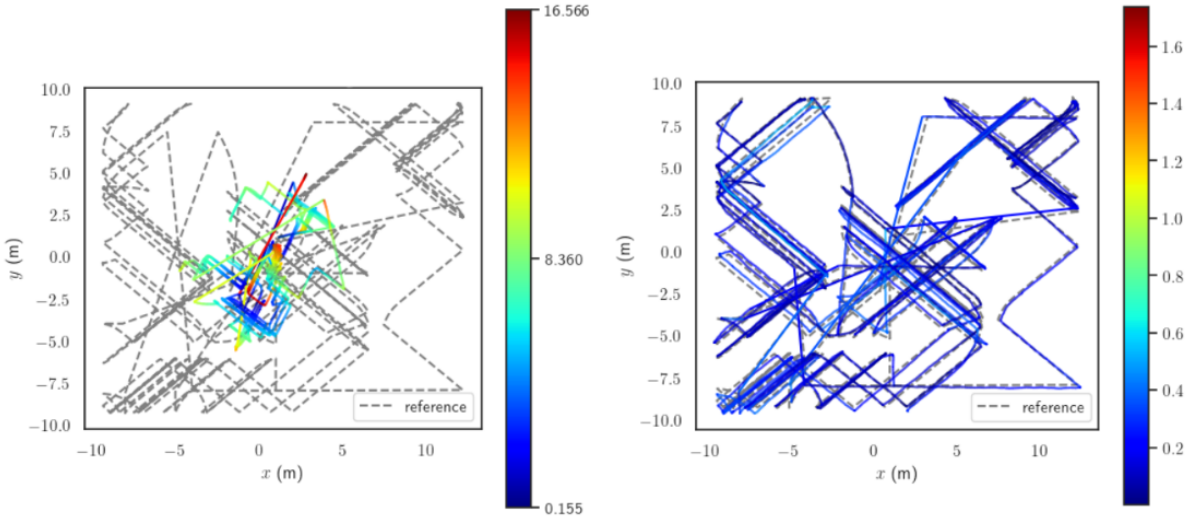

Figure 4.6: Comparison between the optimized pose graph with five Byzantine robots injecting noise into the loop closures without using our protection mechanism (left) and the same scenario with the protection mechanism enabled (right). The results show that the Byzantine fault-tolerant protocol improves pose graph accuracy, as measured by the absolute trajectory error between the trajectories estimated by Swarm-SLAM and the ground truth. The color indicates the magnitude of error at each point as usual.

I want to highlight all the good results of having this protection mechanism with a series of graphs taken from [50] where we see all the metrics of interest and the good behavour of the protection mechanism (Figure 4.7).

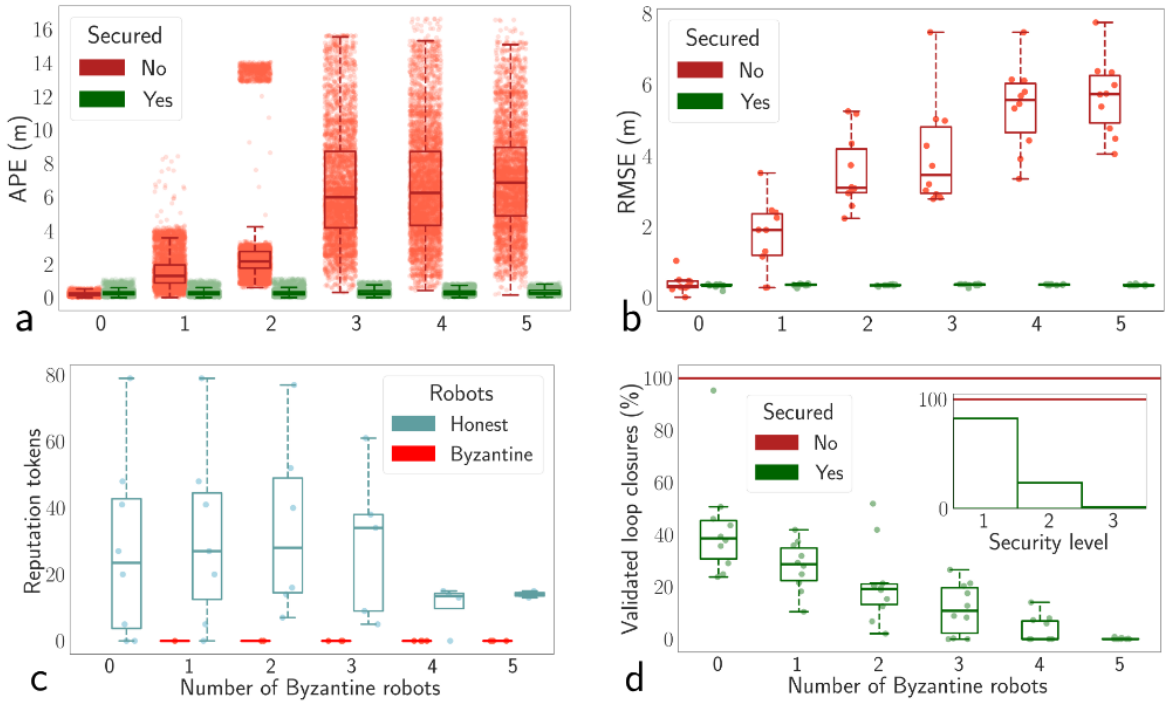

Figure 4.7: (a-b) Comparison of the error in the aggregated robot trajectories after PGO for different numbers of Byzantine robots (x-axis) in a swarm of 8 robots using the original Swarm-SLAM (unsecured) or the Byzantine fault tolerant Swarm-SLAM (secured with smart contract). The boxplots in (a) show the ATE for one representative simulation experiment while in (b) the RMSE is computed over 10 simulation runs (in the same environment but with different starting conditions). (c) Reputation tokens at the end (minute 40) of one representative experiment for Byzantines and non-Byzantine robots. The red boxes (on the left of each green box) are always flat at zero. (d) Proportion of validated loop closures that are used in the Swarm-SLAM's PGO (results for 10 simulation runs for each condition). The red line indicates that 100% of the proposed loop closures are used in PGO in the original Swarm-SLAM. In all panels, we show both the raw data (individual points) and the aggregated data distribution as boxplots.

A final note on this study concerns the **security and efficiency trade-off**. Our protection mechanism enhances the security of the Swarm-SLAM front-end but introduces higher latency, as loop closures must pass through a filtering stage before being incorporated into the pose graph for optimization. Furthermore, increasing the security level (set to 1 by default) results in even greater delays, meaning that greater safety comes at the cost of longer wait times for the final result. This trade-off is important to consider, especially in contexts where system speed is critical, such as safety-critical applications.

4.2. Protecting Swarm-SLAM back-end

This section presents my contributions on protecting Swarm-SLAM's back-end from the issues introduced in Section 3.5 using blockchain technology. I then describe a reputation mechanism developed to safeguard the back-end against Byzantine robots injecting significant noise into their odometry estimates.

4.2.1. Attempting to protect Swarm-SLAM from internal parameter corruption

In Swarm-SLAM, a single elected robot is responsible for performing PGO. As discussed earlier, this creates a single point of failure: if the elected robot behaves as a Byzantine robot, it can corrupt the map, and no security mechanisms are available to verify the correctness of the pose graph. A natural solution to this problem is to decentralize the PGO by extending the process to multiple robots, allowing them to cross-check the consistency of their solutions. The approach I propose takes inspiration from the Byzantine fault-tolerant protocol designed to ensure the correct injection of loop closures into the front-end described in Section 3.3, and is divided into three stages:

1. every time the PGO timer is triggered, each robot selected to perform PGO requests the local pose graphs from the other robots and aggregates them into a single global pose graph.
2. after the aggregation is completed, the robots perform PGO to obtain the optimized pose graphs and send them to the blockchain via a transaction.
3. when the blockchain receives the transactions, a smart contract is triggered to check whether the pose graphs are identical. If they match, the optimized pose graph is accepted and the optimized trajectories are distributed back to all the robots.

The limitation of this solution is that a pose graph can become very large, especially as the number of robots increases—a typical scenario in swarm robotics. Consequently, transmitting such large pose graphs to the Toychain becomes impractical, since the platform was designed for lightweight transactions. Sending heavy pose graph data may introduce long delays, negatively affecting the overall system performance.

In order to solve this problem, I employed a cryptographic primitive called the **hash function**. An hash function is a mathematical algorithm that can take any data as input and produces a fixed-size output called the hash (an alphanumeric code). The hash function has some nice key properties:

- **Determinism:** The same input always produces the same output.
- **Fixed output length:** Regardless of the input size, the output length of the hash is constant.
- **Pre-image resistance:** It is computationally infeasible to determine the input given only its hash.
- **Collision resistance:** It is extremely unlikely for two different inputs to produce the same hash.
- **Avalanche effect:** A small change in the input results in a drastically different hash.
- **Efficiency:** The hash can be computed quickly, even for large inputs.

We could leverage this nice features to protect the back-end of Swarm-SLAM from Byzantine agents. In my approach, the Swarm-SLAM's back-end was modified by decentralizing the PGO, allowing multiple robots to perform it simultaneously. With this approach, multiple robots are selected to perform the PGO. Each of them generates a hash of the resulting pose graph after optimization and submits this hash to the blockchain through a transaction, thereby triggering a smart contract that verifies whether all the submitted hashes match. Assuming that the PGO process is deterministic, if the selected robots are honest, their hashes will match and the resulting pose graph can be accepted. Conversely, if a Byzantine robot is present, the hashes will differ, and the corresponding pose graphs will be rejected. In this way we have a consistency check before publishing the optimized trajectories of the robots. In figure 4.8 I make a visual example of this mechanism.

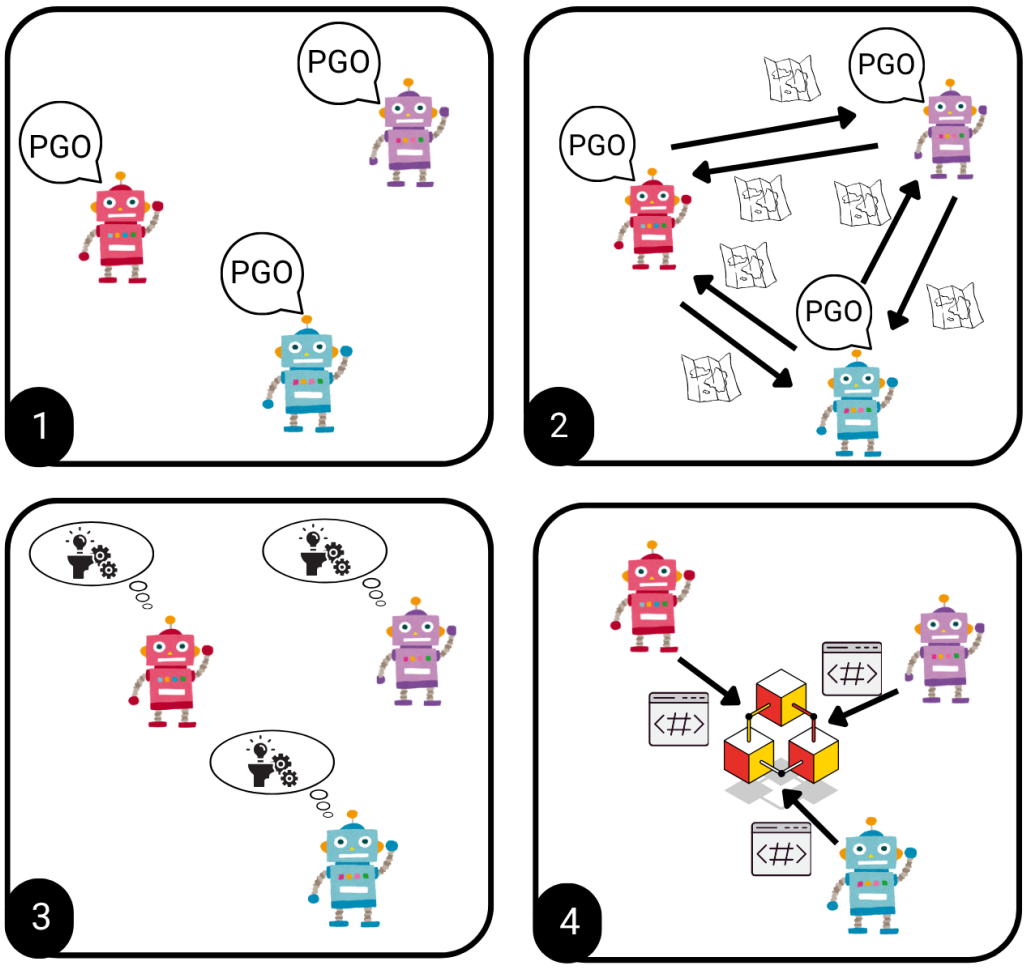

Figure 4.8: Example of a decentralized Swarm-SLAM back-end with three robots performing PGO. Robots first exchange local maps and pose graphs, then each runs PGO and generates a hash of the optimized pose graph. Finally, they submit the hashes to the blockchain, where a smart contract verifies their matching.

In order to apply this protocol, it is necessary to verify the determinism of the PGO process, meaning that the same input should consistently produce the same output. Without this property, the protocol would be ineffective, as the generated hashes would always differ. To assess the determinism of the PGO process, I compare the resulting pose graphs across multiple runs of a single robot performing Swarm-SLAM's PGO, and likewise from three robots operating in parallel, each executing its own PGO. The simulations are based on one representative map taken from the GRACO dataset [80]. The results are shown in Figures 4.9 and 4.10 and show that determinism could not be guaranteed: sometimes the optimized pose graphs overlapped perfectly, while other times they did not, indicating the presence of stochasticity in the system.

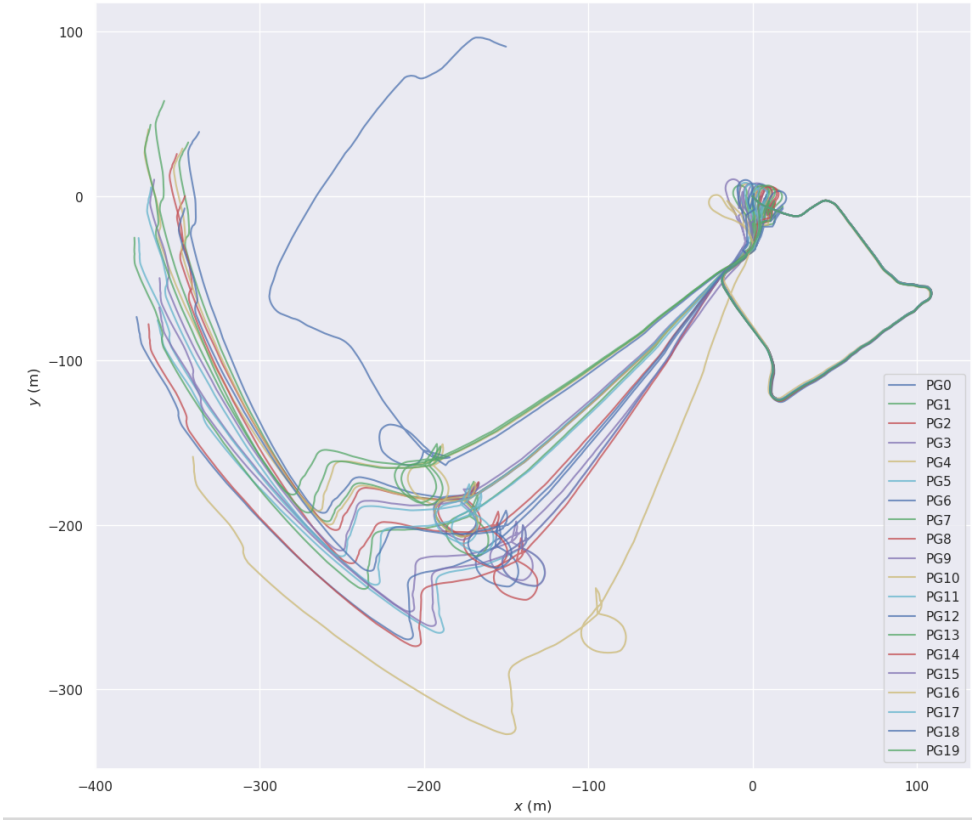

Figure 4.9: Results obtained from running Swarm-SLAM twenty times using the GRACO dataset. As shown, despite using the same input data, the outcomes differ in every run.

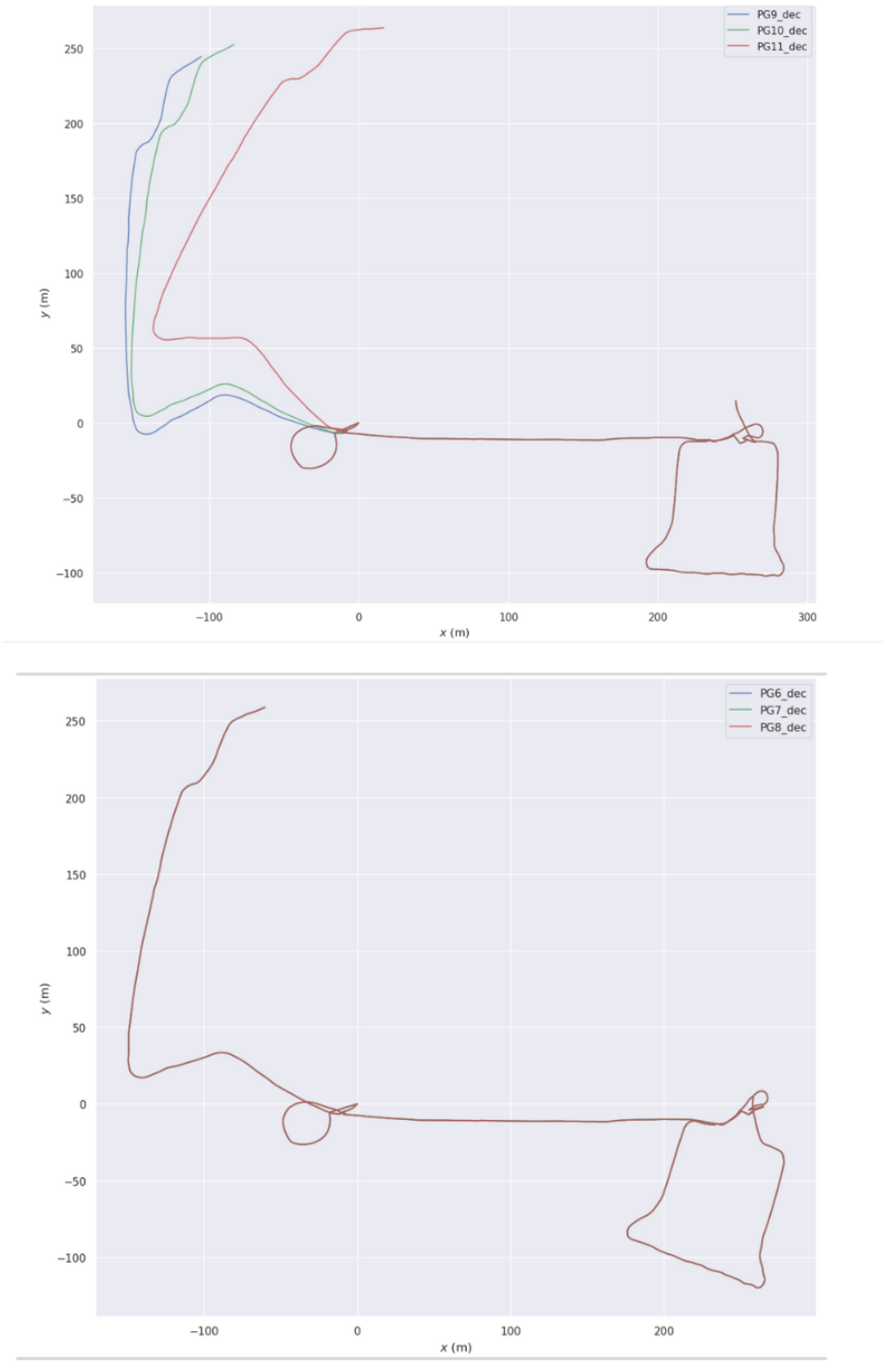

Figure 4.10: Two examples of decentralized PGO outcomes obtained by running Swarm-SLAM with the GRACO dataset. The bottom image shows the ideal case where the three pose graphs perfectly coincide. However, in most cases, the result resembles the upper image, where the three trajectories deviate from each other at certain points.

This protocol could have helped address the various types of inconsistency described in the previous section, and it remains a promising solution for variants of C-SLAM with deterministic PGO processes. Unfortunately, in the case of Swarm-SLAM, the inherent non-determinism of PGO makes the approach unsuitable, which led me to focus instead on developing a reputation mechanism based on PGO-related parameters.

4.2.2. Protecting Swarm-SLAM from noisy or incorrect input data

The covariance matrix, as explained in Subsection 3.5.2, is an important parameter that essentially assigns a weight to each odometry measurement and loop closure during PGO, based on the level of trust in those measurements. This implies that one possible way to neutralize the effect of a Byzantine robot in the system is to increase its covariance matrix values (thereby reducing trust). In other words, the covariance can be interpreted as a measure of a robot's reputation within the system and can be tuned in order to give a robot more or less trust. Since we already have a working solution to handle Byzantine loop closures, as described in Section 4.1, my attention focused on odometry. In this context, a Byzantine robot is one that injects significant noise into its odometry estimates.

In order to test this idea, I created an ARGoS simulation involving five e-puck robots [25]. Four of them are honest robots with correct odometry estimates, while one is Byzantine and injects noise into its odometry estimates. The type of noise injected can be chosen by the user to match the different kinds of noise used in [49]. For example, the noise can be sampled from either a Gaussian or a uniform distribution. Also the number of robots is a user chosen parameters. It is important to mention that loop closures are generated when two robots are within communication range. In the ARGoS simulator, the exact positions of the agents are used to exchange the loop closures and the minimum distance required for them to become peers is a tunable parameters.

A challenging aspect of using ARGoS is that it does not provide ROS2 integration, so I could not directly use Swarm-SLAM, which requires ROS2 to operate. Nevertheless, ARGoS [58] was designed as a flexible simulator for running experiments with large numbers of robots, making it well-suited for swarm robotics research. Since it is not possible to run Swarm-SLAM ROS2 nodes on ARGoS, I had to isolate the essential components needed to design a reputation mechanism. One fundamental component is the GTSAM library [15] and its GNC optimizer [33]. Using GTSAM, I implemented a function that computes both a global optimization error—indicating how well the optimized solution

satisfies the constraints derived from loop closures—and an **individual error** for each robot, reflecting its specific contribution to the global error. Through these individual errors, it is possible to determine whether a robot has faulty odometry: a higher individual error corresponds to a larger odometry error, as the GNC optimizer struggles to refine the map in the presence of inaccurate odometry data. It is important to mention that in a g2o pose graph, we do not have the covariance matrix values directly, but rather their inverse, called the **information matrix**. High values in the information matrix indicate greater trust, and vice versa.

In my work, I designed a reputation mechanism that takes as input the individual optimization error of each robot after every PGO. When the error is low, the robot is rewarded by increasing the corresponding entries of its information matrix, so that its data is given more weight. Conversely, when the error is high, the robot is penalized by decreasing those values, thereby reducing the trust placed in its data. The updated weights are then used in the next PGO iteration. All robots start from the same initial values, which in my case are 100—the same values adopted in Swarm-SLAM. The logic for rewarding and penalizing is customizable and controls how quickly the algorithm converges to the maximum covariance value for Byzantine robots and the minimum covariance value for honest robots. Both the minimum and maximum covariance values are also customizable. Moreover, I implemented a feature that increasingly penalizes a robot as long as it continues to have the highest individual error. The workflow is shown in Figure 4.11. It is important to highlight that the output of the ARGoS simulation is not a pose graph encoded in the g2o format used by the GNC optimizer. Therefore, the output data must be modified to obtain this format, and my project handles this conversion process.

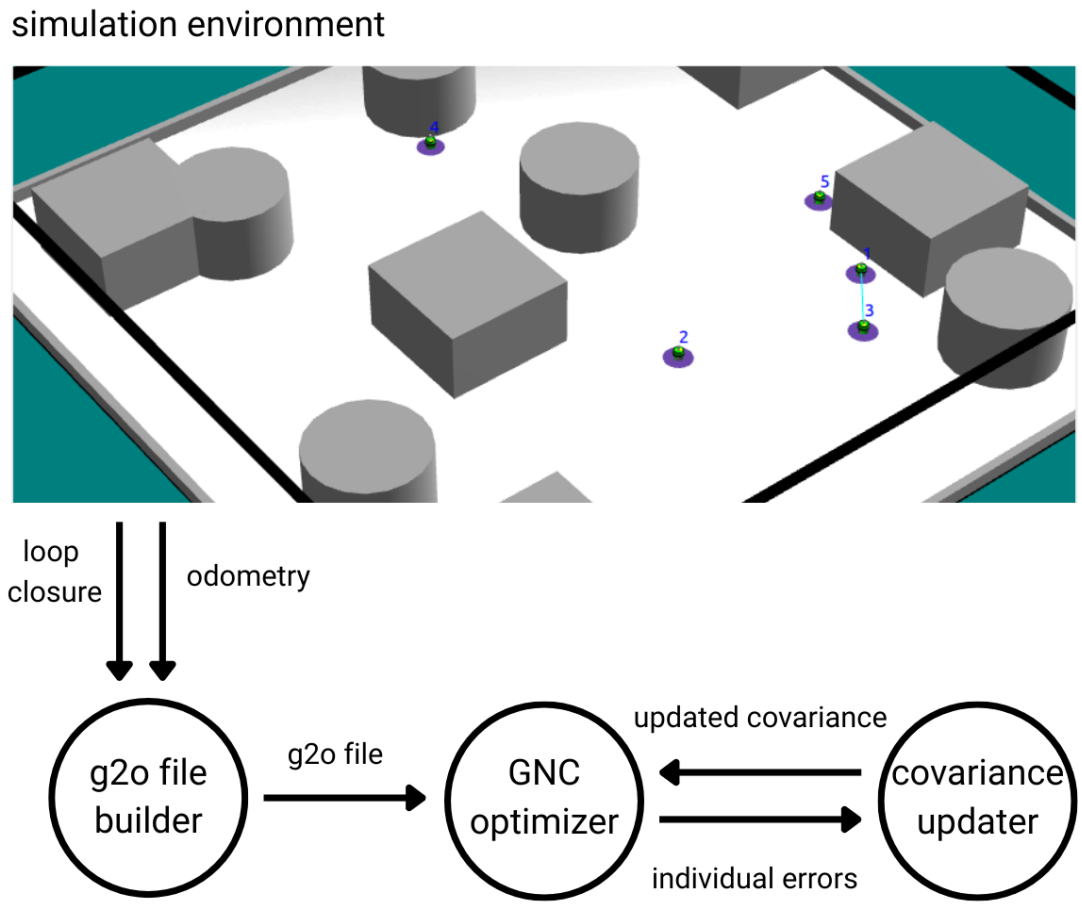

Figure 4.11: Data flow during the execution of the reputation mechanism. Loop closures and odometry from ARGoS are collected into a g2o file for optimization. After optimization, a continuous feedback loop occurs between the optimizer and the covariance updater.

I evaluated the impact of this reputation mechanism by comparing the Absolute Trajectory Error (ATE) and the Root Mean Square Error (RMSE) in scenarios with and without it. In both cases, the ATE and RMSE were computed with respect to the ground truth trajectories, allowing a direct assessment of how the reputation mechanism influences the accuracy of the optimized trajectories. The tests were conducted in ARGoS using five robots, one of which was Byzantine, over 35 minutes long experiments. The timer controlling the interval between two consecutive pose graph optimizations is a customizable parameter; in my experiments, the optimization was executed 30 seconds after the previous one was completed. After each optimization, the information matrix values of each robot were updated based on our trust in them, derived from their individual optimization errors. Employing our covariance-based reputation mechanism results in lower ATE and RMSE compared to the solution without it, demonstrating that the mechanism is effective and that covariance (or information matrix) can indeed be interpreted as a reputation metric and the whole system acts like a protection protocol. I also tracked the progression of the individual errors and the evolution of the information matrix values for each robot throughout the experiment. The information matrix is the inverse of the covariance matrix. This means that if a robot has high covariance values (indicating low trust), it will have low information matrix values, and conversely, if a robot has low covariance values (indicating high trust), it will have high information matrix values. I tracked the information matrix values because they are encoded in the g2o file, but both matrices provide the same information regarding the trust that can be assigned to a robot. As expected, the individual error of the Byzantine robot grows significantly more than that of the others. Consequently, its information matrix values remain consistently low, while those of the other robots increase, indicating a higher level of trust. Figures 4.12, 4.13, 4.14, and 4.15 illustrate the results, showing the mean and standard deviation over 10 runs of the experiment for the previously described metrics: individual optimization errors, information matrix values, mean ATE, and RMSE ATE.

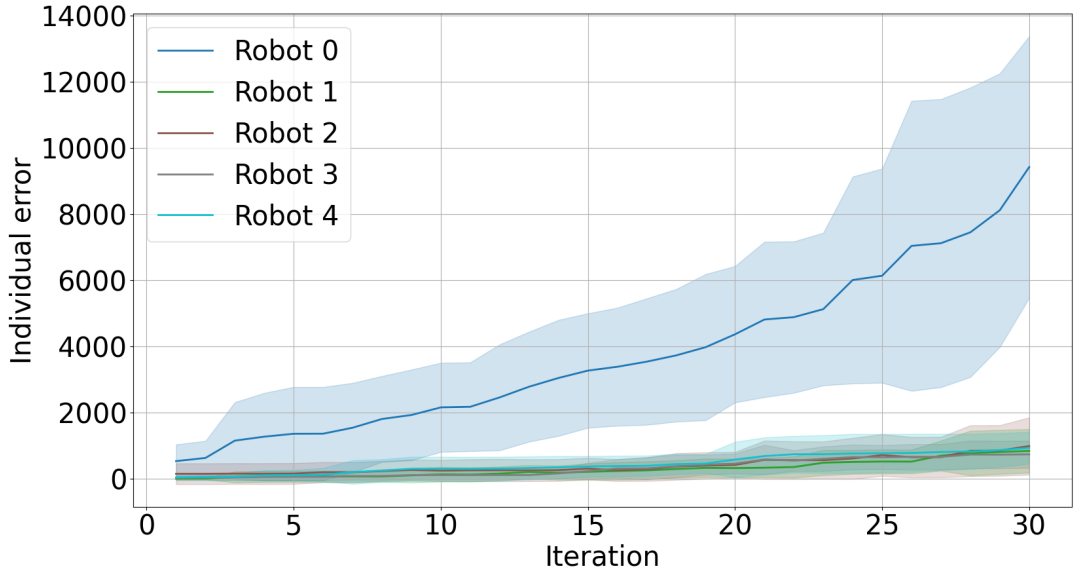

Figure 4.12: Evolution of the individual optimization error for each robot, where Robot0 (in blue) is Byzantine and the others are honest. The y-axis represents the individual error magnitude, while the x-axis shows the number of PGO iterations. The individual optimization error of the Byzantine robot increases significantly more than that of the others, indicating that the optimizer cannot effectively satisfy the constraints imposed by the loop closures due to its noisy odometry. The information matrix values will be updated in the next PGO iteration according to the magnitude of these individual optimization errors. The lines represent the mean values calculated over ten runs of the experiment, while the colored regions around the lines indicate the standard deviation.

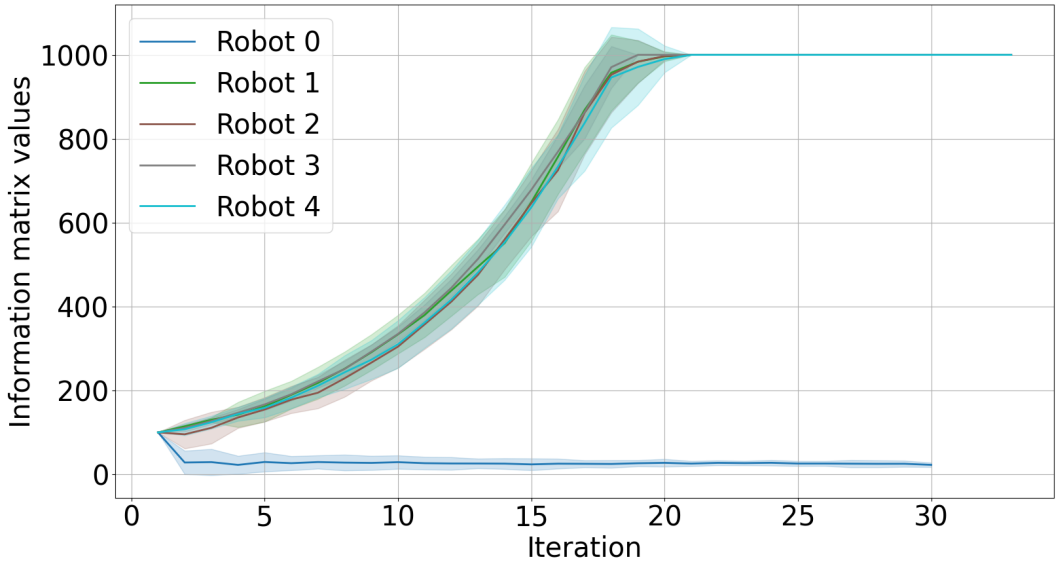

Figure 4.13: Evolution of the information matrix values (inverse of the covariance) for each robot. Robots with high individual errors, such as the Byzantine robot (blue line), are penalized through lower information matrix values, while honest robots with low errors see their values increase. Over iterations, the Byzantine robot's values remain low, whereas the honest robot's values rise and reach the maximum (in this case 1000), indicating high trust in their odometry data. Each robot starts with an information matrix value of 100, which is the fixed value used by Swarm-SLAM and after each PGO iteration, these values are updated. The lines represent the mean values calculated over ten runs of the experiment, while the colored regions around the lines indicate the standard deviation.

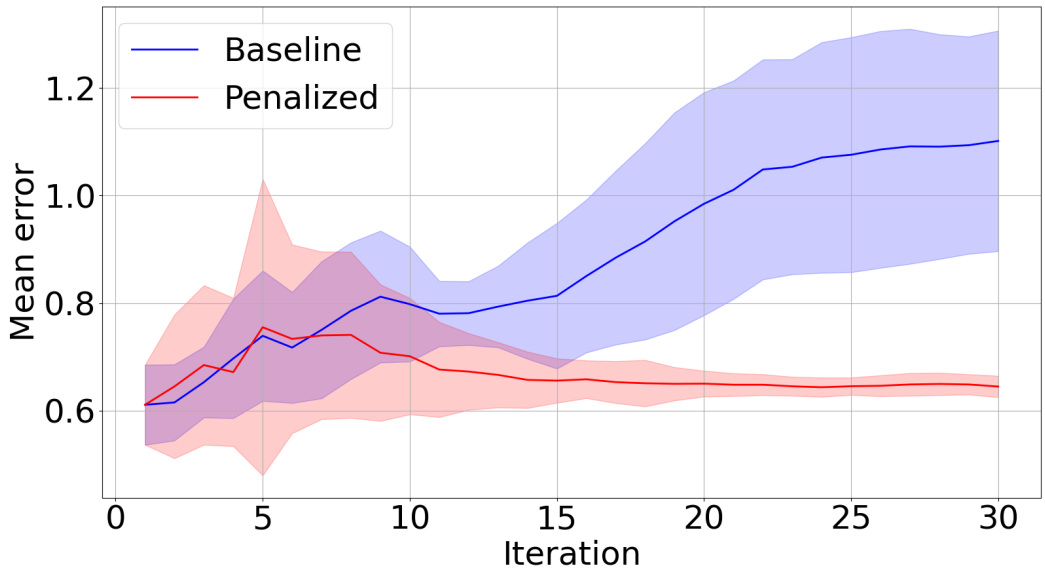

Figure 4.14: This image shows the comparison between the average values of the Mean Absolute Trajectory Error (ATE) for two different solutions against the ground truth. The blue line represents the average ATE of the baseline solution, that is, without any protection mechanism, while the orange line represents the average ATE of the penalized solution, which is the same solution with an active protection mechanism. From the data, it can be observed that the error of the baseline solution tends to increase significantly over the iterations, reaching much higher values compared to the protected solution, which instead maintains a low and stable error over time. This highlights that the protection mechanism applied in the penalized solution is effective in keeping the trajectory error more contained and stable compared to the solution without protection. The colored regions around the lines indicate the standard deviation. All data were collected over ten runs of the experiment.

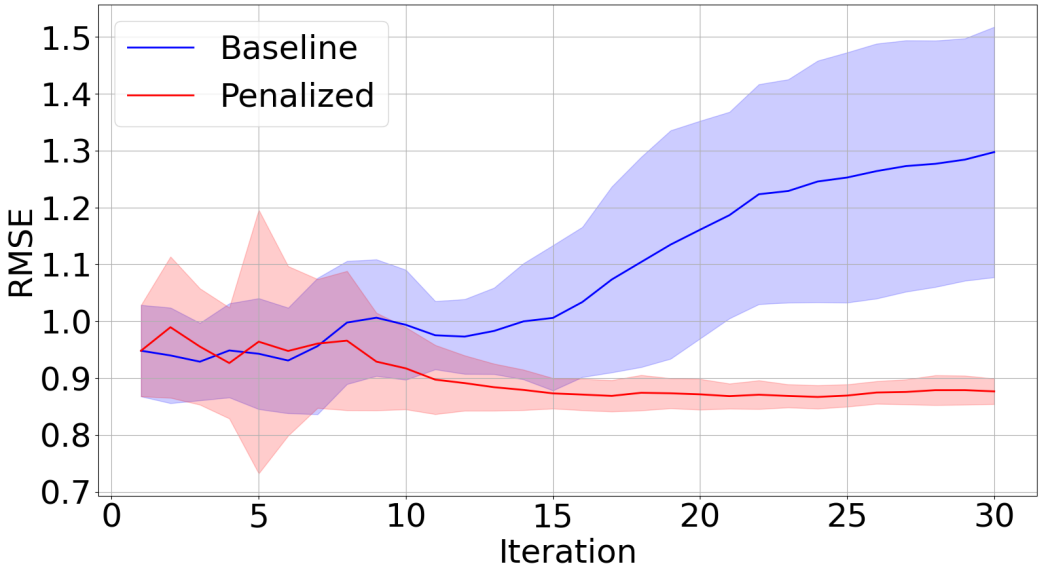

Figure 4.15: This image shows the comparison between the average values of the root mean square Absolute Trajectory Error (RMSE ATE) for two different solutions against the ground truth. The blue line represents the average ATE of the baseline solution, that is, without any protection mechanism, while the orange line represents the average ATE of the penalized solution, which is the same solution with an active protection mechanism. From the data, it can be observed that the error of the baseline solution tends to increase significantly over the iterations, reaching much higher values compared to the protected solution, which instead maintains a low and stable error over time. This highlights that the protection mechanism applied in the penalized solution is effective in keeping the trajectory error more contained and stable compared to the solution without protection. The colored regions around the lines indicate the standard deviation. All data were collected over ten runs of the experiment.

5 | Challenges of the real world implementation

In order to bridge the gap between simulation and real-world implementation, my research focused on the challenges of a physical implementation of Swarm-SLAM, with the aim of testing our Byzantine-fault-tolerant protocol on real robots. Due to hardware limitations, full experiments could not be carried out, but the study highlights the importance of addressing the simulation-to-reality gap. This gap refers to the discrepancies that arise when moving from simulation to real-world experiments, often related to factors ignored in simulation, such as the limited battery life of robots or the changing environmental conditions. Ideally, such experiments should be performed on low-cost robots rather than expensive platforms, in order to ensure scalability and practicality.

As mentioned in the previous chapters, a key feature of Swarm-SLAM is its flexibility, which allows experiments to be conducted with different types of sensors. In my study, I employed the TurtleBot4, a robot equipped with sensors suitable for running Swarm-SLAM.

5.1. The TurtleBot4

The TurtleBot4 [11] is a mobile robot designed for education and research, it is equipped with:

- **OAK-D-PRO camera:** the OAK-D-PRO is a stereo depth camera equipped with an integrated wide field-of-view RGB sensor and infrared laser dot projector, enabling robust depth perception even in low-light conditions. It combines onboard AI processing with real-time depth estimation, making it suitable for robotics applications such as SLAM, navigation, and object detection.
- **RPLIDAR-A1:** the RPLIDAR-A1 is a low-cost 360-degree 2D laser scanner commonly used in robotics for mapping and navigation. It provides a scanning range of up to 12 meters with a sampling rate suitable for SLAM and obstacle avoidance in

indoor environments.

- **IMU:** an Inertial Measurement Unit (IMU) is a sensor that measures linear acceleration and angular velocity using accelerometers and gyroscopes, often combined with a magnetometer. In robotics, IMUs are commonly used for estimating orientation, motion tracking, and sensor fusion in SLAM and navigation tasks.
- **Wheel encoders:** wheel encoders are sensors attached to a robot's wheels that measure their rotation, allowing estimation of distance traveled and velocity. They are commonly used in odometry to support localization and navigation tasks.

There are additional sensors available, but I have described only those that are most relevant for a SLAM application. All onboard sensors are available as ROS topics via the network-connected ROS 2 API. As a computing unit TurtleBots4 are equipped with a Raspberry PI 4 as shown in Figure 5.1. Figure 5.2 shows one of the TurtleBot4 robots available at the University of Konstanz.

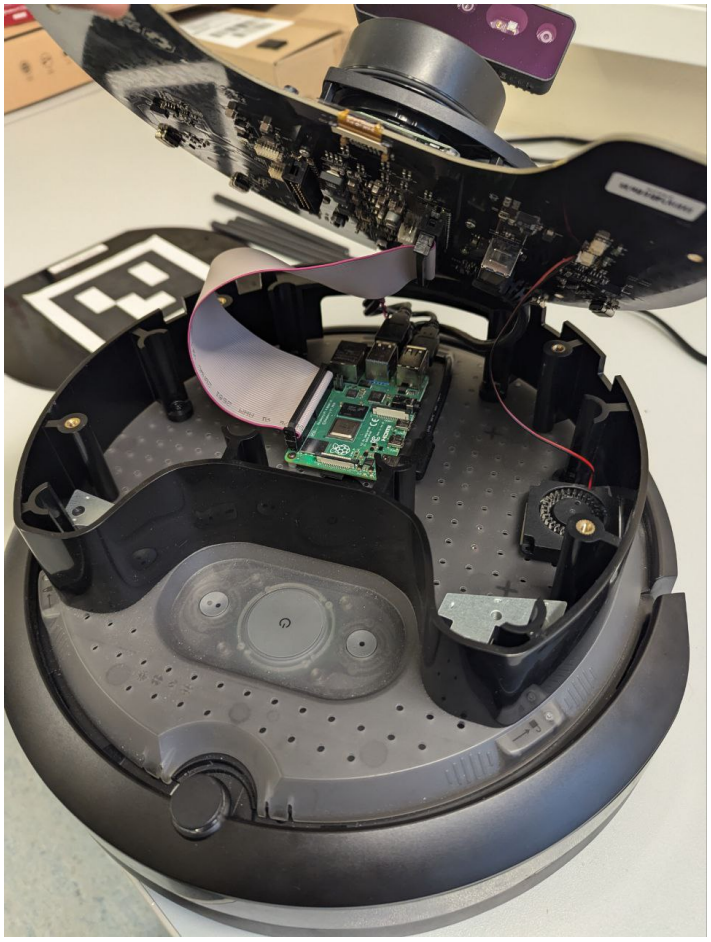

Figure 5.1: Internal view of the TurtleBot4 showing its processing unit, a Raspberry Pi 4, which handles computation for camera and LiDAR.

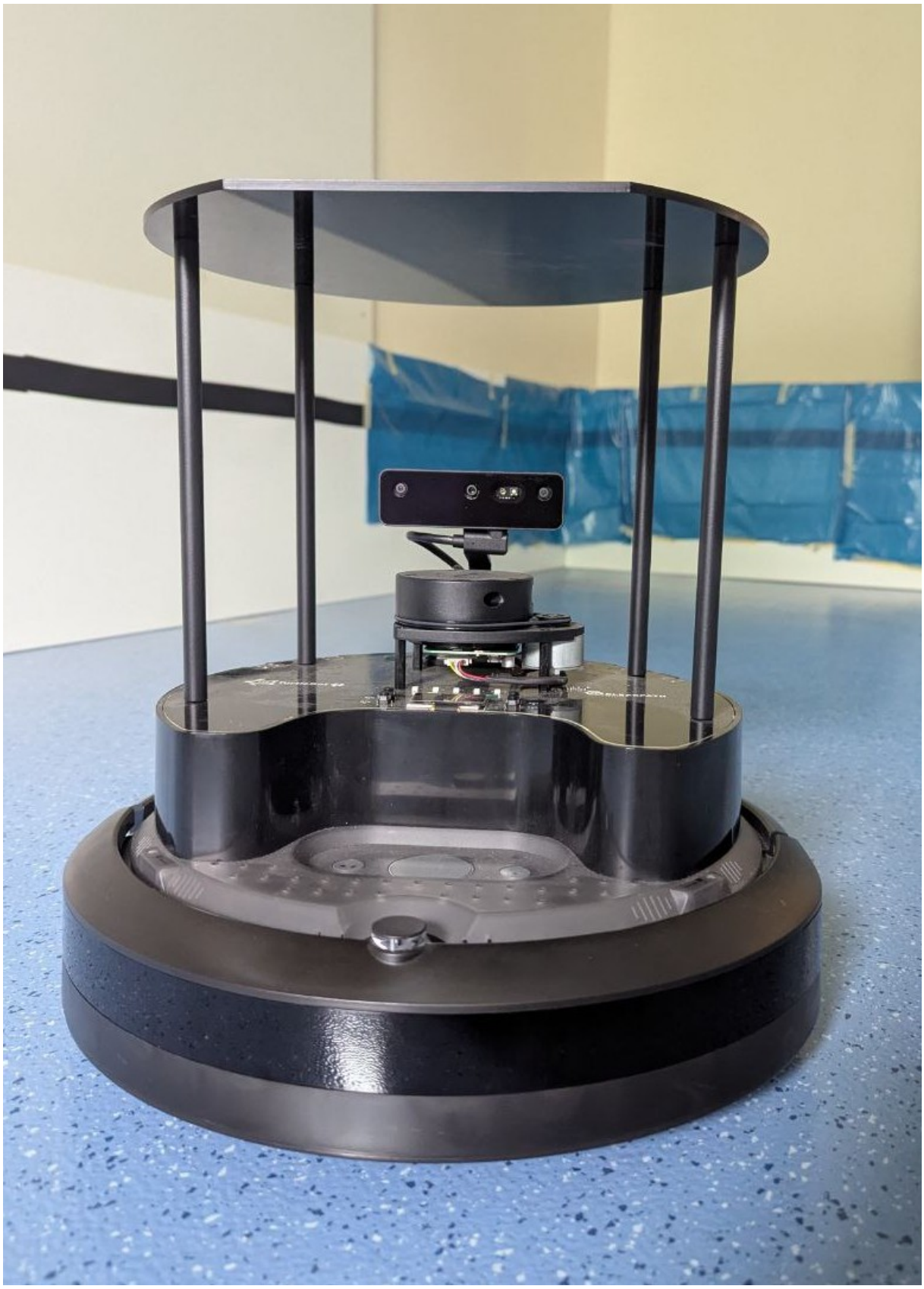

Figure 5.2: One of the TurtleBot4 robots available at the University of Konstanz, used in this study. Multiple units like this one were available for experimentation, equipped with sensors suitable for SLAM applications.

5.2. Data requirements for a real world implementation of Swarm-SLAM

Swarm-SLAM requires an external odometry source as input, and each odometry measurement must be associated with a keyframe obtained from either an RGB-D camera or a 3D LiDAR. Since the TurtleBot4 is equipped with a 2D LiDAR, which is unsuitable for Swarm-SLAM, I opted to use the RGB-D camera to record the keyframes.

I had two choices for the external odometry source:

- **Wheel odometry:** TurtleBot4 robots are equipped with wheel encoders that measure the distance traveled. Encoder readings are published through a built-in ROS 2 topic, natively supported by the TurtleBot4.
- **Visual odometry:** Visual odometry is a technique used to estimate the motion of a robot or camera in 3D space by analyzing sequences of images. In other words, it computes the trajectory of the robot by comparing consecutive frames and determining changes in position and orientation.

If wheel odometry is used, the camera clock must be synchronized with the wheel clock, since the camera and wheels are two separate modules on the TurtleBot, each managed by its own clock. In contrast, when using visual odometry, the odometry estimates and keyframes are automatically synchronized, as they are produced by the same module. From a computational point of view, wheel odometry is lighter and natively available on the TurtleBot. Visual odometry, on the other hand, requires additional software, such as RTAB-Map [39].

At the beginning of my work, I employed RTAB-Map as the visual odometry software in order to simplify the setup and avoid the need to synchronize the different clocks of the TurtleBot. RTAB-Map provides ROS 2 integration, which means it requires input data published as ROS 2 topics. The required ROS 2 topics are:

- **/Camera/Camera_info:** It provides the intrinsic and extrinsic calibration parameters of a camera and contains the metadata required to correctly interpret the images.
- **/Camera/color/image_raw:** It provides the raw RGB image stream captured by the camera without any post-processing or compression.
- **/Camera/depth/image_raw:** It provides the depth map of the environment, where each pixel encodes the distance between the camera and the observed surface.

This information is fundamental for 3D perception tasks, as it allows algorithms to directly associate RGB data with depth values to reconstruct the geometry of the environment

It is important to note that, in order for RTAB-Map to correctly accept and process the input images, the RGB and depth images must have the same resolution. In addition, they must follow specific encoding formats, such as BGR8 for color images and 16UC1 for depth images. In Table 5.1 I summarize all the formats accepted by RTAB-map.

Image type	**Accepted formats**
Color images	`mono8`, `mono16`, `RGB8`, `BGR8`, `BGRA8`, `RGBA8`
Depth images	`32FC1`, `16UC1`, `mono16`

Table 5.1: Accepted image formats in RTAB-Map for color and depth images.

By default, the TurtleBot4 does not publish a depth image topic, as this would require additional computational resources. However, depth images are essential for running visual odometry and SLAM algorithms, which rely on accurate 3D perception. To enable this functionality, I modified the camera launch configuration and replaced the default driver with the one provided by DepthAI [46]. This driver exposes additional ROS 2 topics, including the depth image stream, making the TurtleBot4 fully compatible with the requirements of my experiments.

The last important factor for successfully running RTAB-Map is the frame rate (FPS). A high FPS generally leads to more accurate odometry estimates, while a low FPS can degrade performance. RTAB-Map provides a parameter called odom_quality that indicates the accuracy of the odometry estimates. However, if the FPS is too high for a weak computational unit, odom_quality can drop to zero, meaning the software is no longer able to estimate the robot's trajectory. Similarly, an insufficient FPS can also result in odom_quality dropping to zero. The main challenge is finding a good trade-off between the computational capabilities of the processing unit and the optimal settings for achieving accurate odometry estimates.

Once all these aspects are properly addressed, RTAB-Map runs successfully and provides the /odom ROS 2 topic as input to Swarm-SLAM. The same images fed into RTAB-Map are also used by the Swarm-SLAM front end to associate each odometry measurement with a keyframe and to perform local and global matching for detecting candidate loop closures. The back end, instead, relies only on the odometry messages and the loop closures detected by the front end to build the pose graph. The overall workflow is summarized in Figure 5.3.

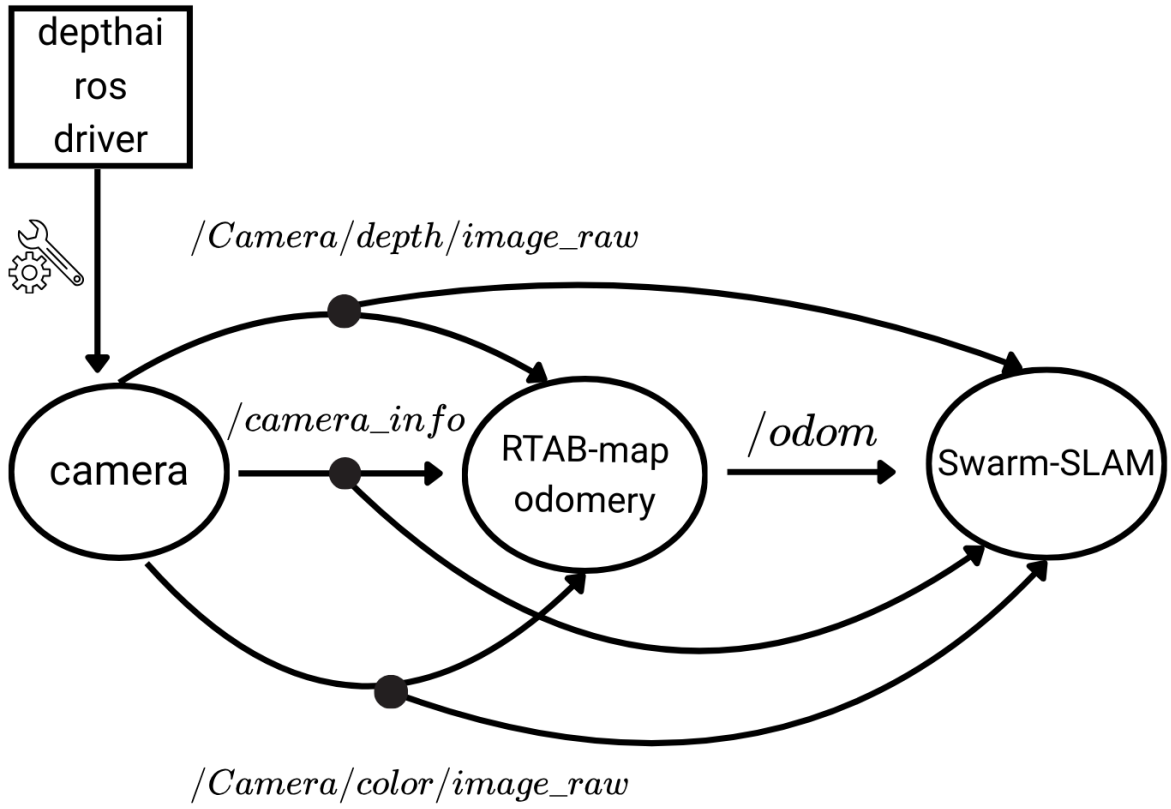

Figure 5.3: Data flow during the execution of a real-world Swarm-SLAM experiment. Starting from the upper left, the selected DepthAI ROS driver is launched to start the camera and provide the three ROS2 image topics required by RTAB-Map and Swarm-SLAM. RTAB-Map receives only the three image topics as input and outputs the odometry estimate. Swarm-SLAM, instead, uses both the three image topics and the odometry estimates in its front end to associate each odometry estimate with a keyframe and to perform local and global matching for detecting candidate loop closures, while its back end uses only the odometry and the detected loop closures to build the pose graph. This kind of graph is also called an rqt graph and is widely used by ROS2 developers to check the flow of data between nodes.

5.3. Hardware requirements for a real world implementation of Swarm-SLAM

As mentioned in previous chapters, a fundamental aspect of swarm robotics is scalability, meaning that the system should continue to behave correctly even as the number of agents increases. This also sets a limit on the hardware complexity of each agent and on the related costs. For this reason, most experiments in swarm robotics are carried out using simple and inexpensive robots, such as the e-puck [25] or Thymio [48].

The authors of Swarm-SLAM carried out real-world experiments in a parking lot to evaluate its viability on resource-constrained platforms. Achieving successful SLAM results in such settings is particularly challenging, as environments with few distinctive features make reliable mapping and localization more difficult. In order to perform these experiments they employed three different robots (Boston Dynamics Spot, Agilex Scout and Agilex scout mini) all equipped with:

- **NVIDIA Jetson AGX Xavier**
- **Intel Realsense D455 camera**
- **Ouster LiDAR OS0-64**
- **VectorNav VN100 IMU**

Figure 5.4: The three robots used by the authors of Swarm-SLAM for the real-world experiments, Figure taken from [40].

They used LiDARs and IMUs for odometry, and RGB-D cameras for inter-robot loop closure detection. However, all these components are expensive pieces of hardware mounted on costly robots, which makes the system difficult to scale from an economic point of view. Given these considerations, my research focused on evaluating the feasibility of Swarm-SLAM using more affordable robots, such as the TurtleBot4.

The TurtleBot4 is equipped with a Raspberry Pi 4, which features a quad-core ARM Cortex-A72 processor, up to 8 GB of RAM, and a set of standard interfaces such as USB 3.0, Gigabit Ethernet, HDMI, and GPIO pins for hardware interaction. During my experiments, however, I observed that the Raspberry Pi 4 was not able to run RTAB-Map as a visual odometry module together with Swarm-SLAM. The temperature of the Raspberry Pi 4 quickly increased to high values, the odom_quality parameter—an internal RTAB-Map metric that estimates odometry accuracy—dropped rapidly to zero, and as a result the Swarm-SLAM framework failed to detect any loop closures. This occurred even when the TurtleBot was moving very slowly in a feature-rich environment. After many tests, I discovered that this bottleneck was caused by CosPlace, the deep learning model used by the front end for visual place recognition. Since it is based on a convolutional neural network that generates embedding vectors to represent the visual content of an image, it requires a GPU to achieve efficient performance. Running it only on the CPU of the Raspberry Pi 4 resulted in very poor performance. Even when decreasing the computational load on the Raspberry Pi 4—for example, by using the /odom ROS2 topic from the wheel encoders instead of running RTAB-Map—the performance did not improve.

Given these results, I performed some experiments connecting the TurtleBot4 camera to my personal laptop, equipped with an Intel i7 processor and an NVIDIA GeForce GTX 960M GPU. The performance improved significantly: the visual odometry ran properly, and the Swarm-SLAM front end was able to detect some intra-robot loop closures. Figure 5.5 shows the setup with the laptop connected to the TurtleBot's camera and placed on top of the robot.

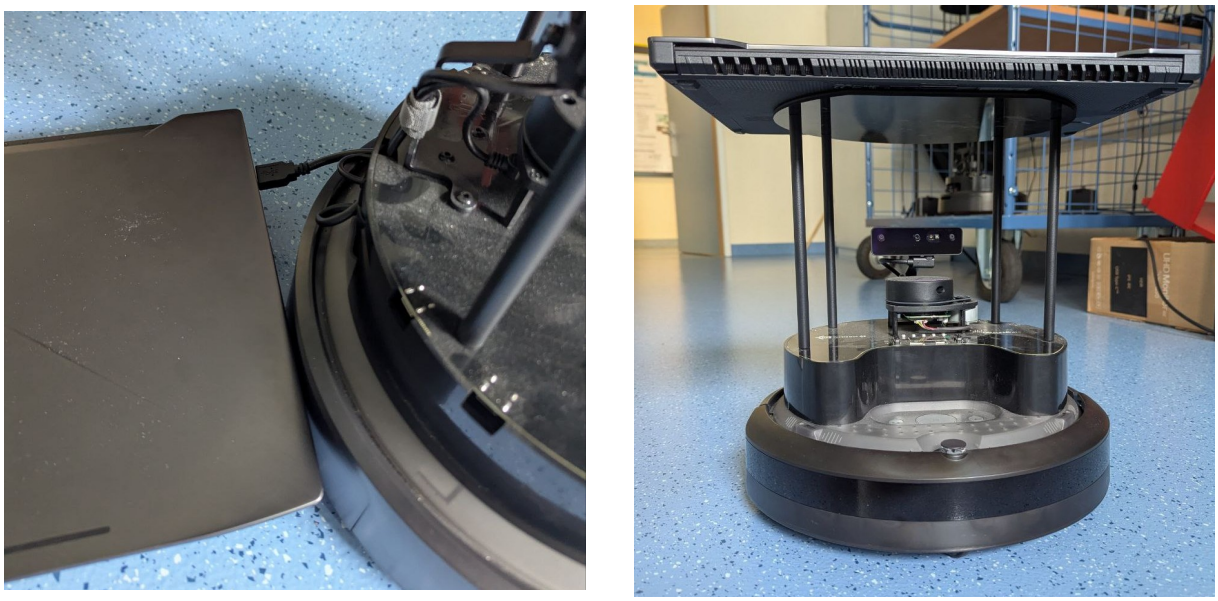

Figure 5.5: On the left, the TurtleBot4 camera is connected to the laptop via USB 3.0 SuperSpeed. On the right, the experimental setup is shown with the laptop placed on the top plate of the TurtleBot4.

I was unable to perform real-world experiments with multiple robots due to the unavailability of GPUs. To test our Byzantine-fault-tolerant protocol, at least four GPUs would have been required to connect to the TurtleBot4 cameras. Nevertheless, my analysis remains relevant for researchers aiming to use Swarm-SLAM for their specific applications, as all the low-level hardware requirements and practical considerations described in this chapter are not detailed in the Swarm-SLAM documentation. My analysis also indicates that Swarm-SLAM cannot be applied on low-cost, resource-constrained devices such as the TurtleBot4; running it requires powerful processing units. This further implies that Swarm-SLAM is not economically scalable.

6 | Conclusion and future developments

In this thesis, I investigated the robustness of a state-of-the-art collaborative SLAM framework in the presence of Byzantine robots. Building on an existing Byzantine-fault-tolerant protocol that addressed malicious behavior in the Swarm-SLAM front end through blockchain technology, I extended the study to the back end, proposing tentative solutions that employ both blockchain and a reputation mechanism leveraging pose graph optimization features. Furthermore, I examined the requirements for successfully running a visual odometry system (RTAB-Map) alongside Swarm-SLAM on real robots, with a focus on potential bottlenecks faced when using low-cost, resource-constrained robots.

My analysis showed that a highly noisy odometry coming from a Byzantine robot can lead to an inaccurate global pose graph, even after pose graph optimization where loop closures are continuously used to refine the map. However, with the proposed reputation mechanism, it is possible to identify the Byzantine robot within the swarm and limit its influence during pose graph optimization, thereby improving the overall map accuracy when compared against the actual robot trajectories (the ground truth).

It is important to acknowledge the limitations of this study. The aim of this thesis was to demonstrate that blockchain technology, already shown to be an effective tool for limiting Byzantine behavior in swarm robotics, can also enhance security in Swarm-SLAM. Moreover, even when blockchain cannot directly mitigate malicious behavior, intrinsic parameters of the Swarm-SLAM framework can be exploited to penalize or reward robots based on the quality of their contributions. However, these tests were conducted in a simplified simulation setup. A promising direction for future work would be to conduct the first real-world experiments of the protected Swarm-SLAM, provided that appropriate hardware is available. Alternatively, another interesting line of research could explore whether recently published collaborative SLAM frameworks exhibit similar behavior to Swarm-SLAM under low hardware requirements, and assess the feasibility of using them as substitutes.

Regarding the limitations of the blockchain implementation, the framework relies on the ToyChain, a simple and highly customizable blockchain introduced in previous work, which allows the coding of smart contracts. It follows the same basic principles as real blockchains, but platforms such as Ethereum are far more complex, particularly in terms of cryptographic security protocols. For the purpose of that work, and consequently for my study, cryptographically secure structures were not essential, since the experiments mainly served as a proof of concept. However, for practical applications of a blockchain-secured Swarm-SLAM system, a widely adopted blockchain platform should be used. Given the promising results already reported with Ethereum in swarm robotics [71], where it has been shown to effectively manage computational requirements in robot swarms, a valuable extension of this work would be the development of an Ethereum-based Swarm-SLAM framework.

Another interesting line of research could be to extend the Byzantine-fault-tolerant Swarm-SLAM front-end in order to identify and neutralize possible colluding robots that can cooperate to validate wrong triangles. Moreover one could expand the covariance-based reputation mechanism described in Chapter 4 performing tests with multiple Byzantine robots.

In conclusion, this thesis has contributed to a deeper understanding of security vulnerabilities in C-SLAM systems by analyzing a state-of-the-art framework and proposing solutions to two critical issues: the creation of inter-robot loop closures by Byzantine robots and the impact of noisy odometry generated by malicious agents. The results not only bring to light challenges that have so far remained unaddressed in C-SLAM, but also provide practical insights for the development of secured Swarm-SLAM implementations. Looking ahead, it is crucial to design secure and robust architectures to advance SLAM systems and effectively tackle the complex challenges posed by Byzantine behavior in robot swarms.

Bibliography

[1] R. Amsters and P. Slaets. *Turtlebot 3 as a Robotics Education Platform*, pages 170–181. 01 2020. doi: 10.1007/978-3-030-26945-6_16.

[2] N. Ayache and O. Faugeras. Building, registrating, and fusing noisy visual maps. *International Journal of Robotic Research*, 7:45–65, 1988. doi: 10.1177/027836498800700605.

[3] G. Beni. The concept of cellular robotic system. In *Proceedings IEEE International Symposium on Intelligent Control 1988*, pages 57–62, 1988. doi: 10.1109/ISIC.1988.65405.

[4] G. Berton, C. Masone, and B. Caputo. Rethinking visual geo-localization for large-scale applications. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4868–4878, 2022. doi: 10.1109/CVPR52688.2022.00483.

[5] J. Bird, J. Blumenkamp, and A. Prorok. Dvm-slam: Decentralized visual monocular simultaneous localization and mapping for multi-agent systems. pages 1–7, 05 2025. doi: 10.1109/ICRA55743.2025.11127510.

[6] J.-L. Blanco-Claraco, A. Leanza, and G. Reina. A general framework for modeling and dynamic simulation of multibody systems using factor graphs. *Nonlinear Dynamics*, 105(3):2031–2053, 2021. doi: 10.1007/s11071-021-06731-6. URL `http://dx.doi.org/10.1007/s11071-021-06731-6`.

[7] C. Campos, R. Elvira, J. J. G. Rodriguez, J. M. M. Montiel, and J. D. Tardos. ORB-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021. doi: 10.1109/tro.2021.3075644. URL `http://dx.doi.org/10.1109/TRO.2021.3075644`.

[8] Y. Chang, K. Ebadi, C. Denniston, M. Ginting, A. Rosinol, A. Reinke, M. Palieri, J. Shi, A. Chatterjee, B. Morrell, A.-a. Agha-mohammadi, and L. Carlone. Lamp 2.0: A robust multi-robot slam system for operation in challenging large-scale underground environments. *IEEE Robotics and Automation Letters*, 7:1–8, 10 2022. doi: 10.1109/LRA.2022.3191204.

[9] W. Chen, X. Wang, S. Gao, G. Shang, C. Zhou, Z. Li, C. Xu, and K. Hu. Overview of multi-robot collaborative SLAM from the perspective of data fusion. *Machines*, 11(6), 2023. doi: 10.3390/machines11060653. URL https://www.mdpi.com/2075-1702/11/6/653.

[10] T. Cieslewski, S. Choudhary, and D. Scaramuzza. Data-efficient decentralized visual SLAM, 2017. URL https://arxiv.org/abs/1710.05772.

[11] Clearpath Robotics. Turtlebot 4. https://clearpathrobotics.com/turtlebot-4/, 2025. Accessed: 2025-08-11.

[12] A. Cramariuc, L. Bernreiter, F. Tschopp, M. Fehr, V. Reijgwart, J. Nieto, R. Siegwart, and C. Cadena. maplab 2.0 – a modular and multi-modal mapping framework. *IEEE Robotics and Automation Letters*, 8(2):520–527, 2023. doi: 10.1109/lra.2022.3227865. URL http://dx.doi.org/10.1109/LRA.2022.3227865.

[13] J. Crowley. World modeling and position estimation for a mobile robot using ultrasonic ranging. In *Proceedings, 1989 International Conference on Robotics and Automation*, pages 674–680 vol.2, 1989. doi: 10.1109/ROBOT.1989.100062.

[14] K. Danilchenko, S. S. Idan, H. Jerbi, and H. Daltrophe. Mrc: a medical-record chain system based on blockchain. *Blockchain: Research and Applications*, page 100344, 2025. ISSN 2096-7209. doi: https://doi.org/10.1016/j.bcra.2025.100344. URL https://www.sciencedirect.com/science/article/pii/S2096720925000715.

[15] F. Dellaert and G. Contributors. borglab/gtsam, 2022. URL https://github.com/borglab/gtsam).

[16] M. Di Pierro. What is the blockchain? *Computing in Science Engineering*, 19(5): 92–95, 2017. doi: 10.1109/MCSE.2017.3421554.

[17] K. J. Doherty, D. M. Rosen, and J. J. Leonard. Spectral measurement sparsification for pose-graph slam. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 01–08, 2022. doi: 10.1109/IROS47612.2022.9981584.

[18] M. Dorigo, G. Theraulaz, and V. Trianni. Swarm robotics: Past, present, and future. *Proceedings of the IEEE*, 109:1152–1165, 07 2021. doi: 10.1109/JPROC.2021.3072740.

[19] M. Dorigo, A. Pacheco, A. Reina, and V. Strobel. Blockchain technology for mobile multi-robot systems. *Nature Reviews Electrical Engineering*, 1(4):264–

274, 2024. doi: 10.1038/s44287-024-00034-9. URL https://doi.org/10.1038/s44287-024-00034-9.

[20] H. Duan, M. Huo, and Y. Fan. From animal collective behaviors to swarm robotic cooperation. *National Science Review*, 10, 2023. doi: 10.1093/nsr/nwad040.

[21] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006. doi: 10.1109/MRA.2006.1638022.

[22] K. Ebadi, M. Palieri, S. Wood, C. Padgett, and A.-a. Agha-mohammadi. Dare-slam: Degeneracy-aware and resilient loop closing in perceptually-degraded environments. *Journal of Intelligent Robotic Systems*, 102, 05 2021. doi: 10.1007/s10846-021-01362-w.

[23] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 834–849, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10605-2.

[24] T. Fukuda, T. Ueyama, Y. Kawauchi, and F. Arai. Concept of cellular robotic system (cebot) and basic strategies for its realization. *Computers Electrical Engineering*, 18 (1):11–39, 1992. doi: https://doi.org/10.1016/0045-7906(92)90029-D. URL https://www.sciencedirect.com/science/article/pii/004579069290029D.

[25] P. Gonçalves, P. Torres, C. Alves, F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, 1, 2009.

[26] M. Grupp. evo: Python package for the evaluation of odometry and SLAM. https://github.com/MichaelGrupp/evo, 2017.

[27] GTSAM.org. Factor graphs, 2020. URL https://gtsam.org/2020/06/01/factor-graphs.html. Accessed: 2025-08-08.

[28] H. Hamann. *Swarm Robotics: A Formal Approach*. 2018. doi: 10.1007/978-3-319-74528-2.

[29] F. Hjálmarsson, G. K. Hreiðarsson, M. Hamdaqa, and G. Hjálmtýsson. Blockchain-based e-voting system. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 983–986, 2018. doi: 10.1109/CLOUD.2018.00151.

[30] Q. Huang. Ethereum: Introduction, expectation, and implementation. *Highlights in Science, Engineering and Technology*, 41:175–182, 2023. doi: 10.54097/hset.v41i.6804.

[31] Y. Huang, T. Shan, F. Chen, and B. Englot. Disco-SLAM: Distributed scan context-enabled multi-robot lidar SLAM with two-stage global-local graph optimization. *IEEE Robotics and Automation Letters*, PP:1–1, 12 2021. doi: 10.1109/LRA.2021.3138156.

[32] IBM. Blockchain for supply chain solutions. `https://www.ibm.com/solutions/blockchain-supply-chain`. Accessed: 2025-09-19.

[33] W. Kang, J. Kim, J. Chung, S. Choi, and T.-W. Kim. Efficient graduated non-convexity for pose graph optimization. In *2024 24th International Conference on Control, Automation and Systems (ICCAS)*, pages 545–548, 2024. doi: 10.23919/ICCAS63016.2024.10773072.

[34] A. Kavitha, C. Sankari, and S. A. Ponmalar. Decentralized digital ledger for secure and transparent crypto voting. In *2025 International Conference on Data Science, Agents Artificial Intelligence (ICDSAAI)*, pages 1–7, 2025. doi: 10.1109/ICDSAAI65575.2025.11011783.

[35] M. Kegeleirs, G. Grisetti, and M. Birattari. Swarm SLAM: Challenges and perspectives. *Frontiers in Robotics and AI*, 8:618268, 2021. doi: 10.3389/frobt.2021.618268.

[36] G. Kim and A. Kim. Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4802–4809, 2018. doi: 10.1109/IROS.2018.8593953.

[37] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004. doi: 10.1109/IROS.2004.1389727.

[38] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011. doi: 10.1109/ICRA.2011.5979949.

[39] M. Labbé and F. Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online opera-

tion. *Journal of Field Robotics*, 36(2):416–446, 2018. doi: 10.1002/rob.21831. URL http://dx.doi.org/10.1002/rob.21831.

[40] P.-Y. Lajoie and G. Beltrame. Swarm-SLAM: Sparse decentralized collaborative simultaneous localization and mapping framework for multi-robot systems. *IEEE Robotics and Automation Letters*, 9(1):475–482, Jan. 2024. doi: 10.1109/lra.2023.3333742. URL http://dx.doi.org/10.1109/LRA.2023.3333742.

[41] P.-Y. Lajoie, S. Hu, G. Beltrame, and L. Carlone. Modeling perceptual aliasing in SLAM via discrete–continuous graphical models. *IEEE Robotics and Automation Letters*, 4(2):1232–1239, 2019. doi: 10.1109/lra.2019.2894852. URL http://dx.doi.org/10.1109/LRA.2019.2894852.

[42] P.-Y. Lajoie, B. Ramtoula, Y. Chang, L. Carlone, and G. Beltrame. DOOR-SLAM: Distributed, online, and outlier resilient SLAM for robotic teams. *IEEE Robotics and Automation Letters*, 5(2):1656–1663, 2020. doi: 10.1109/lra.2020.2967681. URL http://dx.doi.org/10.1109/LRA.2020.2967681.

[43] P.-Y. Lajoie, B. Ramtoula, F. Wu, and G. Beltrame. Towards collaborative simultaneous localization and mapping: A survey of the current research landscape. *Field Robotics*, 2:971–1000, 2022. doi: 10.55417/fr.2022032.

[44] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Langanguages and Systems*, 4, 2002. doi: 10.1145/357172.357176.

[45] M. Lourakis and A. Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software*, 36, 2009.

[46] Luxonis Documentation. Depthai ros driver. https://docs.luxonis.com/software/ros/depthai-ros/driver/, 2025. Accessed: 2025-08-22.

[47] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7 (66), 2022. doi: 10.1126/scirobotics.abm6074. URL http://dx.doi.org/10.1126/scirobotics.abm6074.

[48] F. Mondada, M. Bonani, F. Riedo, M. Briod, L. Pereyre, P. Retornaz, and S. Magnenat. Bringing robotics to formal education: The thymio open-source hardware robot. *IEEE Robotics Automation Magazine*, 24(1):77–85, 2017. doi: 10.1109/MRA.2016.2636372.

[49] A. Moroncelli. Byzantine fault-tolerant Swarm-SLAM through blockchain-based

smart contracts. Master's thesis, Politecnico di Milano, milano, 2023. URL `https://www.politesi.polimi.it/handle/10589/214402`. Accessed: 2025-08-03.

[50] A. Moroncelli, A. Pacheco, V. Strobel, P.-Y. Lajoie, M. Dorigo, and A. Reina. Byzantine fault detection in Swarm-SLAM using blockchain and geometric constraints. In H. Hamann, M. Dorigo, L. Pérez Cáceres, A. Reina, J. Kuckling, T. K. Kaiser, M. Soorati, K. Hasselmann, and E. Buss, editors, *Swarm Intelligence*, pages 42–56, Cham, 2024. Springer Nature Switzerland.

[51] R. Mur-Artal, J. Montiel, and J. Tardos. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31:1147 – 1163, 2015. doi: 10.1109/TRO.2015.2463671.

[52] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 2009.

[53] E. D. Nerurkar, S. I. Roumeliotis, and A. Martinelli. Distributed maximum a posteriori estimation for multi-robot cooperative localization. In *2009 IEEE International Conference on Robotics and Automation*, pages 1402–1409, 2009. doi: 10.1109/ROBOT.2009.5152398.

[54] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. Dtam: Dense tracking and mapping in real-time. In *2011 International Conference on Computer Vision*, pages 2320–2327, 2011. doi: 10.1109/ICCV.2011.6126513.

[55] V. Niculescu, T. Polonelli, M. Magno, and L. Benini. Ultra-lightweight collaborative slam for robot swarms. *IEEE Access*, PP:1–1, 01 2025. doi: 10.1109/ACCESS.2025.3574556.

[56] A. Pacheco, V. Strobel, and M. Dorigo. A blockchain-controlled physical robot swarm communicating via an ad-hoc network. In *Swarm Intelligence – Proceedings of ANTS 2020 – Twelfth International Conference*, volume 12421 of *Lecture Notes in Computer Science*, pages 3–15, Cham, Switzerland, 2020. Springer. doi: 10.1007/978-3-030-60376-2_1.

[57] A. Pacheco, V. Strobel, A. Reina, and M. Dorigo. *Real-Time Coordination of a Foraging Robot Swarm Using Blockchain Smart Contracts*, pages 196–208. 2022. doi: 10.1007/978-3-031-20176-9_16.

[58] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo.

Argos: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6:271–295, 2012. doi: 10.1007/s11721-012-0072-5.

[59] J. Placed, J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. Castellanos. A survey on active simultaneous localization and mapping: State of the art and new frontiers. *IEEE Transactions on Robotics*, PP:1–20, 2023. doi: 10.1109/TRO.2023.3248510.

[60] A. Reina. Robot teams stay safe with blockchains. *Nature Machine Intelligence*, 2(5):240–241, 2020. doi: 10.1038/s42256-020-0178-1. URL `https://doi.org/10.1038/s42256-020-0178-1`.

[61] A. Reina, E. Ferrante, and G. Valentini. Collective decision-making in living and artificial systems: editorial. *Swarm Intelligence*, 15(1):1–6, 2021. doi: 10.1007/s11721-021-00195-5. URL `https://doi.org/10.1007/s11721-021-00195-5`.

[62] D. Rodríguez-Losada, F. Matia, and A. Jimenez. Local maps fusion for real time multirobot indoor simultaneous localization and mapping. volume 2, pages 1308 – 1313 Vol.2, 2004. doi: 10.1109/ROBOT.2004.1308005.

[63] S. Saeedi, M. Trentini, M. Seto, and H. Li. Multiple-robot simultaneous localization and mapping: A review. *Journal of Field Robotics*, 33:3–46, 2016.

[64] P. Schmuck and M. Chli. CCM-SLAM: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams. *Journal of Field Robotics*, 36:763–781, 12 2018. doi: 10.1002/rob.21854.

[65] P. Schmuck, T. Ziegler, M. Karrer, J. Perraudin, and M. Chli. Covins: Visual-inertial SLAM for centralized collaboration. In *2021 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 171–176, 2021. doi: 10.1109/ISMAR-Adjunct54149.2021.00043.

[66] S. Se, H. Ng, P. Jasiobedzki, and T.-J. Moyung. Vision based modeling and localization for planetary exploration rovers. In *Proceedings of the 55th International Astronautical Congress*, volume 12. International Astronautical Federation, 2004.

[67] X. Shao. Review on VSLAM based on deep learning. *Theoretical and Natural Science*, 41:94–100, 2024. doi: 10.54254/2753-8818/41/2024CH0187.

[68] R. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5, 1987. doi: 10.1177/027836498600500404.

[69] H. Strasdat, J. Montiel, and A. Davison. Editors choice article: Visual SLAM: Why filter? *Image and Vision Computing*, 30:65–77, 2012. doi: 10.1016/j.imavis.2012.02.009.

[70] V. Strobel, E. C. Ferrer, and M. Dorigo. 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 '18)*, pages 541–549, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

[71] V. Strobel, E. Castelló Ferrer, and M. Dorigo. Blockchain technology secures robot swarms: A comparison of consensus protocols and their resilience to Byzantine robots. *Frontiers in Robotics and AI*, Volume 7 - 2020, 2020. doi: 10.3389/frobt.2020.00054. URL `https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2020.00054`.

[72] V. Strobel, A. Pacheco, and M. Dorigo. Robot swarms neutralize harmful Byzantine robots using a blockchain-based token economy. *Science robotics*, 8:eabm4636, 2023. doi: 10.1126/scirobotics.abm4636.

[73] Y. Tian, Y. Chang, F. H. Arias, C. Nieto-Granda, J. P. How, and L. Carlone. Kimera-multi: Robust, distributed, dense metric-semantic SLAM for multi-robot systems, 2021. URL `https://arxiv.org/abs/2106.14386`.

[74] D. Ulysse, P. Alexandre, S. Volker, A. Reina, and D. Marco. Toy-chain. IRIDIA Technical Report Series 9, IRIDIA, the Artificial Intelligence Laboratory at the Université Libre de Bruxelles, 2023.

[75] H. Xu, P. Liu, X. Chen, and S. Shen. D^2slam: Decentralized and distributed collaborative visual-inertial slam system for aerial swarm. *IEEE Transactions on Robotics*, PP:1–20, 01 2024. doi: 10.1109/TRO.2024.3422003.

[76] G.-Z. Yang, J. Bellingham, P. Dupont, P. Fischer, L. Floridi, R. Full, N. Jacobstein, V. Kumar, M. McNutt, R. Merrifield, B. Nelson, B. Scassellati, M. Taddeo, R. Taylor, M. Veloso, Z. Wang, and R. Wood. The grand challenges of science robotics. *Science Robotics*, 3:eaar7650, 01 2018. doi: 10.1126/scirobotics.aar7650.

[77] H. Yang, P. Antonante, V. Tzoumas, and L. Carlone. Graduated non-convexity for robust spatial perception: From non-minimal solvers to global outlier rejection. *IEEE Robotics and Automation Letters*, 5(2):1127–1134, 2020. doi: 10.1109/lra.2020.2965893. URL `http://dx.doi.org/10.1109/LRA.2020.2965893`.

[78] H. Zhao, A. Pacheco, V. Strobel, A. Reina, X. Liu, G. Dudek, and M. Dorigo. A generic framework for Byzantine-tolerant consensus achievement in robot swarms. pages 8839–8846, 2023. doi: 10.1109/IROS55552.2023.10341423.

[79] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang. An overview of blockchain technology: Architecture, consensus, and future trends. 2017. doi: 10.1109/BigDataCongress.2017.85.

[80] Y. Zhu, Y. Kong, Y. Jie, S. Xu, and H. Cheng. Graco: A multimodal dataset for ground and aerial cooperative localization and mapping. *IEEE Robotics and Automation Letters*, 8(2):966–973, 2023. doi: 10.1109/LRA.2023.3234802.

List of Figures

List of Tables

Acknowledgements

I would like to thank everyone who contributed to this work.

I am deeply grateful to the Centre for the Advanced Study of Collective Behaviour for hosting me. The atmosphere and energy in your research group have been truly inspiring, and I hope you will continue achieving great success. My special thanks go to Dr. Andreagiovanni Reina and Ir. Alexandre Pacheco for their invaluable guidance: you not only helped me grow technically through your advice, but also enriched me personally.

I would also like to thank Prof. Francesco Amigoni, my internal supervisor at Politecnico di Milano, who first introduced me to Dr. Andreagiovanni Reina.

I am profoundly grateful to my family for their endless love and support. I especially thank my parents for standing by me through every step of my academic journey and for being close to me in my darkest moments. My deepest thanks to my brother Nicola, who encouraged me not to give up in the face of difficulties. I am also grateful to my grandmothers, who raised me but are sadly no longer with us, as well as to my uncles, Marta, Jacopo, and Nova—the time spent with you was always a breath of fresh air.

A heartfelt thank you goes to Eleonora, who has been by my side in every moment, supporting me constantly and encouraging me always to give my best.

I also warmly thank my closest friends—Giovanni, Nicoletta, and Matteo—knowing I can always count on you means a lot to me.

My gratitude goes as well to Elberns, my flatmate in Milan. It was an honor to share an apartment with an engineer like you.

I sincerely thank Simea, Melvin, Mariana, Jannik, Luis, and Sophika for making my stay in Konstanz truly special. I will never forget our dinners at home and our swims in the lake.

Finally, I extend my thanks to all my other friends—though I cannot name you all individually, I am grateful for your presence in my life, and to all the wonderful people I had the chance to meet abroad during this journey.