POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Byzantine Fault-Tolerant Swarm-SLAM through Blockchain-based Smart Contracts

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

Author: **Angelo Moroncelli**

Student ID: 971576
Advisor: Prof. Francesco Amigoni
Co-advisors: Dr. Andreagiovanni Reina, Dr. Volker Strobel, Ir. Alexandre Pacheco
Academic Year: 2023–2024

Abstract

In order to navigate and explore unknown environments without external localisation systems, autonomous robotics systems must be able to perform Simultaneous Localisation And Mapping, SLAM for short. Through SLAM techniques, robots can build a map of the environment and localise themselves in it in real time.

Decades of intensive research have led to efficient solutions in single-robot SLAM, however recent attention has shifted to the design of multi-robot SLAM systems where groups of robots cooperatively build a map and improve localisation estimates through exchange of local information. Multi-robot SLAM offers unique opportunities for improved efficiency, robustness, and parallelisation, however facing the challenges of scalability and consistent data aggregation of potentially conflicting pieces of information.

While robustness is often indicated as an intrinsic characteristic of multi-robot systems thanks to the presence of many robots, recent research has shown that robot redundancy and parallelisation of operations are not sufficient to achieve system robustness against misbehaving robots. It is reasonable to assume that a subset of robots may misbehave, deviating from the designed algorithm, due to internal errors or to external malicious tampering. In agreement with decentralised system literature, I name such misbehaving robots as Byzantine robots.

This thesis studies the robustness of the state-of-the-art framework for multi-robot SLAM in decentralised robot swarms, which is called Swarm-SLAM. I show that Swarm-SLAM is vulnerable to the presence of Byzantine robots: even one of them is sufficient to jeopardise the entire system. Therefore, inspired by recent research on blockchain-based swarm robotics, I built a security layer for Swarm-SLAM through blockchain-based smart contracts, which are distributed algorithms running on the blockchain. I test my solution with a set of physics-driven simulations of groups of eight robots running algorithms coded in ROS2 and a custom blockchain framework. The results show that the proposed blockchain-based solution makes Swarm-SLAM tolerant to relatively large Byzantine faults.

My analysis also shows that the increase in Byzantine fault tolerance is compensated by

a decrease in system efficiency. This thesis discusses such a robustness-efficiency trade-off and also comprehensively discusses the security issues that Swarm-SLAM, in the specific, and multi-robot SLAM, in general, face and how they could be potentially addressed through future research in blockchain-based solutions for robotics.

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

Abstract in lingua italiana

Al fine di navigare ed esplorare ambienti sconosciuti senza l'ausilio di sistemi esterni di localizzazione, i sistemi robotici autonomi devono essere in grado di eseguire la localizzazione e mappatura simultanea, abbreviato in SLAM. Attraverso le tecniche SLAM, i robot possono costruire una mappa dell'ambiente e localizzarsi al suo interno in tempo reale.

Decenni di ricerca intensiva hanno portato a soluzioni efficienti nel campo dello SLAM per robot singoli, tuttavia, recentemente, l'attenzione si è spostata al design di sistemi SLAM multi-robot, in cui gruppi di robot costituiscono in cooperazione una mappa e migliorano le stime di localizzazione attraverso lo scambio di informazioni locali. Lo SLAM multi-robot offre opportunità uniche per migliorare l'efficienza, la robustezza e la parallelizzazione, affrontando tuttavia le sfide di scalabilità e di aggregazione coerente di dati potenzialmente in conflitto.

Mentre la robustezza è spesso indicata come una caratteristica intrinseca dei sistemi multi-robot grazie alla presenza di molti robot, ricerche recenti hanno dimostrato che la ridondanza e la parallelizzazione delle operazioni non sono sufficienti per ottenere robustezza del sistema contro robot malfunzionanti. È ragionevole presumere che un sottoinsieme di robot possa comportarsi in modo errato, deviando dall'algoritmo progettato, a causa di errori interni o di manomissioni esterne. In accordo con la letteratura sui sistemi decentralizzati, chiamo tali robot malfunzionanti "robot bizantini".

Questa tesi studia la robustezza del framework allo stato dell'arte per lo SLAM multi-robot in sciami di robot decentralizzati, chiamato Swarm-SLAM. Mostro che Swarm-SLAM è vulnerabile alla presenza di robot bizantini, anche solo uno di essi è sufficiente per compromettere l'intero sistema. Pertanto, ispirato dalle recenti ricerche sulla robotica degli sciami che utilizza la tecnologia blockchain, ho sviluppato un applicativo di sicurezza per Swarm-SLAM che sfrutta contratti intelligenti, che sono algoritmi distribuiti in esecuzione sulla blockchain. Testo la mia soluzione con un insieme di simulazioni di gruppi di otto robot che eseguono algoritmi codificati in ROS2 e un framework blockchain personalizzato. I risultati mostrano che la soluzione proposta rende Swarm-SLAM tollerante a

guasti bizantini significativi.

La mia analisi mostra anche che l'aumento della tolleranza ai guasti bizantini è correlato a una diminuzione dell'efficienza del sistema. Questa tesi discute tale compromesso tra robustezza ed efficienza e affronta in modo completo le problematiche di sicurezza che Swarm-SLAM, in particolare, e lo SLAM multi-robot, in generale, affrontano e come potrebbero essere potenzialmente affrontate attraverso future ricerche in soluzioni basate su blockchain per la robotica.

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

Acknowledgements

I would like to express my sincere gratitude to all those who have contributed to the success of this work.

First and foremost, I want to express my deepest gratitude to Dr. Andreagiovanni Reina for his insightful guidance. His advice motivated me during these months and helped me to find my path, beyond my Master Thesis research. I also want to thank Dr. Volker Strobel and Ir. Alexandre Pacheco for providing me with the opportunity to delve into these topics. They constantly supported me and really gave me a chance to deepen my knowledge in the field. Thanks for the support I received during this research journey, both technically and personally, which played a pivotal role in the development of this thesis.

Thanks to Ir. Pierre-Yves Lajoie, PhD student from École Polytechnique de Montréal, for collaborating with me and for the constant support and availability. He introduced me to Swarm-SLAM and helped me technically in the usage of the framework he developed.

I also express my gratitude to Prof. Marco Dorigo and all the people from IRIDIA for offering me the incredible opportunity to spend my days in their laboratory. I will always be grateful for the stimulating and welcoming environment I found.

A special thanks goes to Prof. Francesco Amigoni, that has been my internal supervisor at Politecnico di Milano and introduced me to Dr. Andreagiovanni Reina for the first time.

Moreover, I want to sincerely thank my family for their unwavering love. I am deeply grateful for my parents; they have consistently supported me. I extend my thanks to my sister, Chiara. Strong appreciation goes to my grandparents for their constant encouragement and kindness. In particular, I would like to express my gratitude to my grandfather, who, unfortunately, cannot be here.

A big thanks goes to Francesca for sharing this new experience abroad with me and for being a constant source of support. Thank you for always encouraging me to do my best.

Lastly, I extend my appreciation to all my friends. While I cannot name everyone, thanks

to those who have been with me since the beginning, playing a fundamental role in my life, and to the fantastic new people I encountered during this path, with whom I shared this experience.

Contents

1 | Introduction

This Master's thesis was conducted as part of a collaborative effort within Politecnico di Milano and the IRIDIA laboratory at the Université Libre de Bruxelles. I had the privilege of working under the guidance of Prof. Francesco Amigoni, Dr. Andreagiovanni Reina, Dr. Volker Strobel, and Ir. Alexandre Pacheco.

1.1. Problem statement

This work is a research study within the domain of swarm robotics [12] and distributed decision making processes, specifically addressing security challenges in the context of robotic swarms engaged in Simultaneous Localisation And Mapping (SLAM), where robots with Byzantine behaviour are present. A Byzantine robot is a component of the swarm that behaves improperly or maliciously, disrupting the normal functioning of the system.

Swarm-SLAM [28] is a pioneering approach to decentralised SLAM, it employs robot swarms for scalable, flexible, and fault-tolerant exploration and mapping in dynamic, unknown environments. Despite the promising potential, this field is nascent, with limited frameworks and results, particularly concerning security and robustness in the face of Byzantine robots exhibiting negative behaviour with respect to the swarm's mission. A Byzantine fault-tolerant protocol, custom designed for the Swarm-SLAM framework, is proposed.

The research involves an in-depth investigation and analysis of the potential consequences of various types of attacks in a distributed system and proposes novel approaches to mitigate them, leveraging on blockchain, a specific form of distributed ledger technology.

1.2. Background

1.2.1. Simultaneous Localisation and Mapping (SLAM)

SLAM (Simultaneous Localisation And Mapping)—a fundamental capability of autonomous systems—is the process that allows the system (e.g., a robot or an autonomous vehicle) to localise itself in an unknown environment, without external localisation systems (e.g., without GPS), while building a map of its surroundings in real-time. The solution to this problem is seen as the first step towards fully autonomous robots [21].

In several robotic applications, from autonomous vehicles to robots that perform rescue operations in inaccessible environments, reliable SLAM algorithms are a key enabling capability [15]. SLAM is having today a growing success in practical applications, thanks to the development of high-quality and low-cost cameras and LiDAR sensors, jointly with increasing computational capability of embedded systems. An example of the achievements of SLAM in modern research are the recent results in active SLAM [42], robots are able to plan and control their motion to build the most accurate map.
The concept of simultaneous localisation and mapping made its debut in a seminal paper of Smith et al. in 1986 [51]. In their work, Smith et al. introduced an estimation theory-based approach for the localisation and mapping of robots, effectively bridging the realms of robotics and artificial intelligence. Over the course of three decades, SLAM has undergone significant development. In 1988, Ayache and Faugeras [2] conducted pioneering research in visual navigation, while Crowley [9] explored Kalman filter-based visual navigation for mobile robots. While at the beginning bundle adjustment optimisation techniques were considered less due to the problem of high computational demands; in 2009 the concept of sparsity in bundle adjustment [33] emerged, paving the way for graph-optimisation based methods like DTAM [40], LSD-SLAM [16], ORB-SLAM [37], and other remarkable algorithms. Today, with the rapid advancements in artificial intelligence and deep learning, researchers embarked on exploring the integration of deep learning into traditional filtering and graph-optimisation based methods to address the intrinsic challenges of environmental adaptability faced by traditional methods [61].
Consequently, the application of deep learning in tackling SLAM-related issues has become a prominent and evolving trend.

Without loss of generality, it is possible to see a SLAM architecture as a composition of two main submodules: the front end and the back end (see Figure 1.1). Conceptually, they address two distinct problems, but they interplay in a continuous exchange of information while improving the solution.

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.

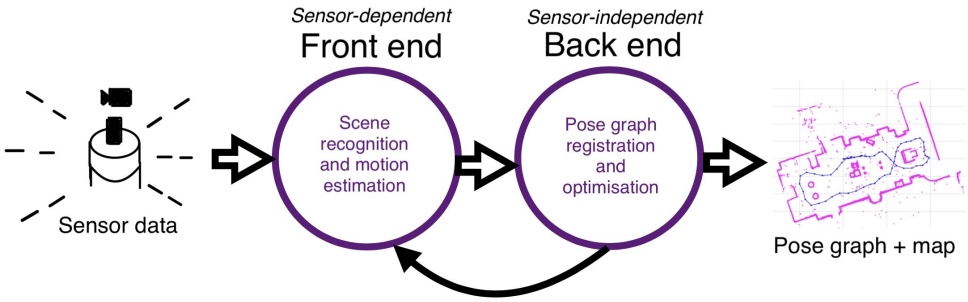

Figure 1.1: Typical information processing flow in SLAM [35].

1.2.2. Swarm robotics and distributed decision making systems

Swarm robotics is an innovative approach to robotics inspired by the collective behaviour of social animals. By following simple rules and engaging in short-range interactions, swarm robotics aims to design and create systems composed of a large number of robots that self-organise to perform tasks cooperatively.
The key characteristics of a robot swarm should be:

1. **Autonomy:** the ability to perform operations and to make decisions without continuous human intervention.
2. **Large number of entities:** the robotic system can be composed of considerable quantities of agents that cooperate to achieve a common goal. This ability requires advanced algorithms and communication systems to optimise the overall system performance.
3. **Limited individual capabilities:** the single entities that compose the swarm of robots usually have simple hardware and software. They are limited to perform simple actions.
4. **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. Scalability is the property of a robotic system to efficiently adapt and expand its capabilities as needed. For example, the team size can vary drastically without requiring a redesign of the overall system.

5. **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 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 introduced in robotics by Beni in 1988 [3]. The author discussed the conceptual basis for the theory and engineering of a new type of robotic system composed of autonomous robotic units which accomplish tasks in cooperation.
Subsequently, Fukuda discussed the possibility to design self-organising robotic systems taking inspiration from biology, introducing the concept of cellular robotics in 1992 [18]; in the same years Beni and Wang introduce the term 'swarm intelligence' as a definition for all those systems in which the collaboration between the individuals leads to emergent patterns that resemble unpredictable and non random behaviours.
In the early stages of swarm robotics, researchers delved into the study of swarming behaviours found in various species such as ants, birds, and fish. Their aim was to understand these behavioural patterns and find ways to replicate them in diverse robotics systems. Furthermore, the research was fuelled by a multitude of inspirations, ranging from the flocking patterns of birds to the communal actions of ant colonies.
Numerous studies [20] [12] [14] and research endeavours sought to emulate swarming habits, including activities like foraging, flocking, stigmergy, and cooperation through direct and indirect communication. Stigmergy, in this context, refers to the indirect communication and coordination observed in species like termites and ants. Foraging refers to the search and collection of resources in the environment and flocking pertains to the coordinated movement of individuals in large groups, these are two typical behaviours observed in animals.
Typically, swarm robotics extracts engineering principles from the study of those natural systems in order to provide multi-robot systems with comparable abilities [12]. The computational modelling approach used to study swarm behaviours extended in the development of successful optimisation algorithms, an example is the Ant Colony Optimisation [11].
Swarm robotics is still a very active area of research and several recent works delved into the mathematical modelling of the complex interaction between robots, analysing the macro-scale effect of individual and local rules. Reina et al. for example, leveraged on tools from dynamical system theory to model the effect that strongly opinionated minorities of agents can have on the collective opinion in large systems [47]. Recently,

these formal approaches were adopted to investigate the properties of opinion selection mechanisms in best-of-n decision making problems, where the issue is to optimally select the best choice from a set of possibilities based on some defined criteria [60] [46].

Swarm robotics is a developing field aimed at creating robust, scalable and flexible systems for various applications. However, as of now, most artificial collective systems have been primarily developed in academic settings and have not yet addressed real-world problems. The test scenarios used to verify the actual collaboration between robots have been mostly limited to foraging and collective movement. Nevertheless, the ambition is to use the collaborative systems observed in nature as abstractions to solve real-world problems.

1.2.3. Blockchain technology

Blockchain technology is a revolutionising paradigm for distributed data management and secure information storage that allows users to exchange economic transactions without relying on a central authority. In a couple of words blockchain is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a network [24].

The first example of this technology was introduced in 2008, by an individual or group operating under the pseudonym "Satoshi Nakamoto", when the first paper appeared with the title "Bitcoin: A Peer-to-Peer Electronic Cash System" [38]. It means that, originally, blockchain was established as a purely decentralised way to exchange money transactions over the Internet through the Bitcoin cryptocurrency. Only in the subsequent years, with the rise of new digital ledgers like Ethereum and many other platforms, blockchain became a new framework for distributed computing allowing users not only to share on the network economical transactions but also to execute computer programs (called smart contracts) on a distributed computing platform.
In particular, the Ethereum protocol with its cryptocurrency (the Ether) is the most widely used blockchain platform worldwide that implements smart contracts, and the Ethereum Virtual Machine is what defines the rules for computing a new valid state from block to block [44].
It operates on the principles of transparency, security and immutability, making it a unique and reliable tool for various applications. Each transaction is grouped into a block, and these blocks are linked together in a chain. Once a transaction is recorded in the blockchain, it becomes extremely challenging to alter it, ensuring data integrity. The structure of a blockchain is depicted in Figure 1.2 and in Figure 1.3.

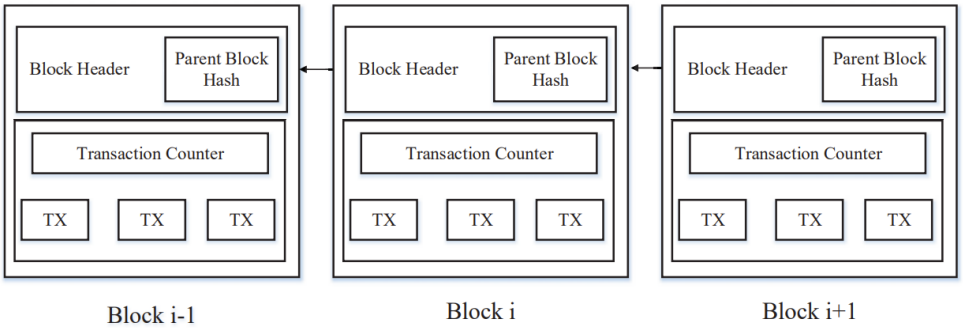

Figure 1.2: Three subsequent blocks of a chain. Figure taken from [63].

This technology extends beyond cryptocurrencies, finding use in supply chain management, voting systems, healthcare records, and in particular in swarm robotics.

Blockchain's decentralised nature eliminates the need for a central authority, and at the same time, it fosters transparency and trust among participants through its consensus protocol. Previous research has shown that decentralised smart contracts executed via blockchain technology can take the role of 'meta-controllers' to achieve a secure consensus in peer-to-peer robotic networks [53].

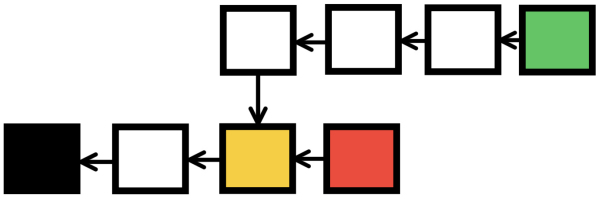

Figure 1.3: Illustration of a chain structure where new blocks are linked to the previous one as usual in a blockchain. However, a fork in the chain occurred at a specific point. The network transitioned to a state where an alternative sequence of blocks started to generate, leading to a new consensus. Consequently, two distinct versions of the chain coexist, represented by the red and green blocks.

Smart contracts

A smart contract is an algorithm (comprising functions and variables) stored and executed on the blockchain. To initiate a smart contract or invoke its functions, individuals must generate a transaction and broadcast it within the blockchain network. The blockchain's nodes are responsible for maintaining and updating the algorithm's internal state, i.e., storing and computing the values of its variable.
The same smart contract is run from every participant in the network once a new transaction is generated and its execution can modify the values of the state variables.

Below, a short example of what was previously discussed is reported from the work of Strobel et al. “Blockchain technology secures robot swarms” [53].

Imagine that an Ethereum smart contract is employed to facilitate the selection of a talent show winner on television. In this setup, the audience has the option to cast their votes for their preferred candidate, such as Alice or Bob, by initiating a transaction (e.g., accompanied by a 0.01 ether contribution) to the television station’s smart contract on the public Ethereum blockchain. The smart contract itself maintains a record of the vote count for each candidate. Furthermore, it defines a programmable condition: when one candidate collects 100’000 votes, a prize of 1’000 ether is automatically transferred to the Ethereum address associated with that candidate.

This example underscores several advantages of smart contracts when compared to traditional voting scenarios:

1. The condition of the contract and the vote counts are fully transparent.
2. Once cast, existing votes remain immutable and cannot be manipulated or discarded.
3. The prize money is guaranteed to be disbursed promptly when the specified condition is met.

The Toy-Chain

While there are currently several blockchain-based smart contract platforms in existence, in my thesis work I have employed a simplified blochchain platform, named as Toy-Chain [57]. The Toy-Chain is a mock-up blockchain module realised in Python and developed at IRIDIA, the Artificial Intelligence Laboratory of the Univerité Libre de Bruxelles in Belgium.

The Toy-Chain abstracts the standard cryptographic mechanisms employed by the blockchain (e.g., the public key cryptography method is not present) and only includes the components that are relevant to study how a blockchain can be used to control the behaviour of robot swarms. The Toy-Chain comprises 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 grouped together in a block and then added to the blockchain. Blocks are formed by a list of transactions and the values of state variables. A block also contains a timestamp, a unique cryptographic hash that univocally identifies it and the hash of the parent's block that links the current to the previous block in the chain. Figure 1.4 depicts an example of the information that is registered into a block.

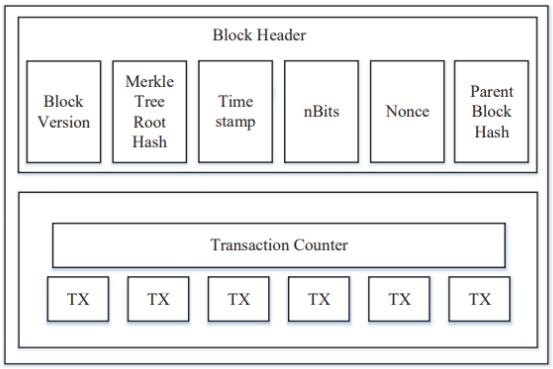

Figure 1.4: Representation the general structure of a single block in a chain. It contains a set of transactions (Tx) and all the other information that are registered into a block [63]. A block in the Toy-Chain is constructed in the same way.

- **TCP and synchronisation**

  The Transmission Control Protocol (TCP) is used in the Toy-Chain 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 protocols**

  The consensus protocol defines the rules for block production and verification. All nodes unanimously commit to adhering to the consensus rules, commencing their operations with the genesis block. The genesis block, being the starting block of a blockchain, is identical for every node, forming the foundational starting point. Its specifics are contingent on the chosen consensus mechanism and are established during the consensus initialisation process. As an example, the genesis block is the black block in Figure 1.3. The Toy-Chain natively includes two alternative consensus protocols: the proof-of-work and the proof-of-authority.

Proof-of-Authority (PoA) is the consensus protocol I adopt to validate the results. The basic idea of POA is that only certain nodes, called authorised signers, can produce blocks. It is a consensus protocol in which reputation is used to decide who is entitled to actively participate in the blockchain network. Network participants with a certain level of reputation and identity are recognised as the authorities or nodes that can produce blocks and achieve consensus.

1.3. State of the art

1.3.1. Multi-robot SLAM and C-SLAM

Robot utilise SLAM to navigate and map unknown environments. However, uncharted environments are extensive, necessitating the collaborative efforts of multiple robots. When a team of robots is deployed, each can explore a portion of the environment and then collaborate by sharing their findings, creating a comprehensive map. The multi-robot collaborative SLAM approach has the capability to address several challenges encountered also by single-robot SLAM schemes. These challenges include mitigating individual costs, minimising global error accumulation, reducing computational load, and dispersing the risk of single-point failure; but achieving this coordination is a complex task [6] [26]. These were the motivations, in the early 2000s, under which researchers realised that the need for collaboration between robots that are simultaneously building a map of their surroundings and localising themselves was a promising paradigm. Alongside the developments in multi-sensor fusion technology, initially, algorithms tailored to single-robot systems were adapted to multi-robot systems, resulting in numerous successful instances. These achievements include the development of effective algorithms like the multi-robot EKF algorithm [48] and the multi-robot cooperative localisation algorithm [39]. In the subsequent years, the utilisation of multi-source data increased and the integration of SLAM technologies with deep learning enhanced its adaptability, reducing the likelihood of failures and pushing forward the research in multi-robot collaborative SLAM [6].

In 2016, Saeedi et al. surveyed the field of multi-robot SLAM [49], realising that optimisation-based methods (compared to filter-based methods) were becoming a standard. In recent years, lots of very promising multi-agent SLAM platforms have become a reality, although most of these architectures are not very scalable to many robots and their capabilities remain limited to few-robots scenarios. Researchers focused on solving fundamental problems that arise in collaborative scenarios, such as robustness to perceptual aliasing on real-time fully distributed systems [56] and facing challenges related to teams of heterogeneous mobile robots [5].

Nowadays, a widely used umbrella term to refer to collaborative systems is Collaborative-SLAM (C-SLAM). Different popular open-source frameworks, that include some of the previously cited works, have been proposed: D-SLAM [7], DOOR-SLAM [30], COVINS [50], Kimera-Multi [56], Disco-SLAM [22], LAMP 2.0 [5], maplab 2.0 [8] and Swarm-SLAM [28].

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

1. **Types of sensors available**, such as stereo cameras (s), LiDAR sensors (l) and RGB-D cameras (d).
2. **Decentralisation**, meaning that decision-making and control are spread out, allowing for more local autonomy and flexibility.
3. **Robustness**. A system that can operate well under uncertainty.
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**, when not all the data or information is available or collected at all times. Instead, the robots gather and process data as they come, extracting key information while ignoring or minimising less important data.

Recently, Lajoie et al. [28] summarised how the existing methods compare with each other in terms of these key properties. I report their analysis in the Table 1.1, emphasising that fully achieving all these desirable features is very difficult, and that the only framework that does so is Swarm-SLAM.

	Sensor	**Decentralised**	**Robust**	**Sporadic**	**Sparse**
DSLAM [7]	s	✓			
DOOR-SLAM [30]	s,l	✓	✓	✓	
COVINS [50]	s		✓		✓
Kimera-Multi [56]	s	✓	✓	✓	
Disco-SLAM [22]	l	✓	✓	✓	
LAMP 2.0 [5]	l		✓		✓
maplab 2.0 [8]	s,d,l		✓		
Swarm-SLAM [28]	s,d,l	✓	✓	✓	✓

Table 1.1: Extensive comparison of C-SLAM open-source frameworks reported in [28].

Parallel to the evolution of SLAM technology, swarm robotics has proven to be promising for exploring vast environments with fully autonomous decentralised robots, all while

maintaining adaptable properties. In 2021, Dorigo et al. reviewed the most promising research directions of this field [45].
Considering SLAM and swarm robotics together has become natural to exploit the advantages of robot swarms in self-organising and redundant systems comprising a potentially high number of entities, that are performing simultaneous localisation and mapping.
In 2021, Kegeleirs et al. reviewed the state of the art in SLAM with robot swarms, a rather novel approach that in some way extends what is done in the multi-robot applications. Although several advancements have been recently made [25] [28], this field still lacks definitions, frameworks, and general results [26].
In particular, Kegeleirs et al. reported that the majority of multi-robot SLAM systems at the time they were writing, referred to centralised architectures, i.e., non-swarm systems. They highlighted how these systems are able to produce accurate maps, however they are typically not scalable, and are prone to failure in hostile circumstances since they cannot easily adapt to rapidly changing environments.

Currently, Swarm-SLAM [28] stands out as a unique framework that tackled all the major challenges presented by swarm robotics in a collaborative SLAM scenario. It is a novel approach to solving map creation and localisation, within a fully distributed swarm in a resource-efficient manner.
Swarm-SLAM allows robots to use different types of sensors, such as stereo cameras, RGB-D cameras, and LiDAR sensors, and to operate with sporadic connectivity and significantly reduced communication demands compared to prior methods. To minimise data exchanges, Swarm-SLAM introduces an original budgeted strategy for identifying potential inter-robot loop closures through algebraic connectivity maximisation, inspired by recent work on pose graph sparsification [10]. Swarm-SLAM achieves results more swiftly and with reduced communication overhead with respect to other C-SLAM systems. The whole architecture is based on the Robot Operating System 2 (ROS2) [34], ensuring compatibility with the majority of robotic systems. They evaluated the overall system's performance conducting real-world experiments.
Even though we saw that flexibility, scalability and decentralisation are achieved in the state-of-the-art works, according to the challenges reported by Kegeleirs et al. [26] we are still far from having fully autonomous reliable and robust SLAM systems with robot swarms. Indeed, when authors refer to robustness properties they indicate the problem of perceptual aliasing [29], since it is of utmost importance and the major cause of failure in SLAM systems. However, when we consider more complex robotic networks, specifically groups of agents with non-ideal behaviour or even potentially able to commit malicious actions, the reliability of the whole system is in danger. Lately, the interest in the study

of security in distributed decision-making systems increased significantly [54], but to the best of my knowledge no one focused on these problems in the SLAM literature, and consequently, there are no solutions to the Byzantine fault-tolerant problem yet. Before Swarm-SLAM applications could be an effective solution to real-world problems, Byzantine fault-tolerance, as well as other hard challenges, have to be addressed, e.g., assuring scalability in large-scale experiments typical of swarm robotics and long-term reliability.

1.3.2. Blockchain secures robot swarms

Building upon the challenges posed by the Byzantine generals problem [32], Strobel et al. [52] were the first to address the scenario where Byzantine robots could adversely influence swarm actions. Byzantine robots are those exhibiting "Byzantine behaviours"—actions deriving from faults or malicious interference. Detecting these behaviours is difficult, and they hold the potential to disrupt the attainment of accurate decisions. Incorrect agreement can impede the swarm from successfully accomplishing common tasks, such as when robots collectively agree on an inaccurate objectives, resulting in wasted resources. Therefore, it is of utmost importance to assess the impact of Byzantine behaviours on achieving collective decisions within robot swarms. In severe cases, incorrect collective decisions can result in catastrophic outcomes, such as unanimous agreement on dangerous actions, a concern supported by more recent works [62].

Secure decentralised generalised transaction ledgers [58] appear to be a potential solution to the problem of security in robot swarms [12], thanks to the possibility of maintaining a decentralised database while preserving privacy with cryptographic protocols.

In [53] and [41] the authors show how blockchain can ensure consensus—the collaborative decision-making process among a group of robots to achieve a common goal or behaviour—in the presence of Byzantine robots. Inspired by Bitcoin [38] and Ethereum [58] they implemented blockchain-based smart contracts to control robots and prevent attacks in a robot swarm that enabled the robots to agree on an estimated ratio between white and black tiles on the experimental floor. They released the interface as open-source software. This approach enhances security and holds potential for robot-to-robot transactions in swarms, among several other applications. Most of the results are tailored to precise experimental setups to solve specific problems, but attempts to propose more generic frameworks based on a blockchain smart contract that enables robot swarms to achieve secure consensus in an arbitrary observation space exist [32].

Detailed experiments and analysis performed on a blockchain-based token economy maintained by 24 physical Pi-puck robots confirm great promises [54]. Building upon the recent promising works that leverage on blockchain technology to secure robot swarms, in this

work I aim to address the Byzantine fault-tolerant problem in systems that perform SLAM with swarm of robots.

1.4. Structure and methodology

The thesis is organised as follows: in this introductory Chapter 1, I provided an overview of the problem contextualising my study in the existing research literature. I presented different concepts that comes from interdisciplinary fields, that a reader should know to fully understand the work.
Chapter 2 aims to deeply describe the robustness problem in C-SLAM systems. I present my case study and I show the main motivations for this work through an example. In Chapter 3, a Byzantine fault-tolerant protocol for C-SLAM applications is described. Chapter 4, presents my findings, followed by a discussion that interprets the results and their implications. Finally, the conclusive Chapter 5 summarises the key messages and offers insights for future research.

My methodology initially focus on the latest C-SLAM framework to comprehend its operation. I test the state-of-the-art software for C-SLAM, I analyse its weaknesses in presence of possible Byzantine individuals in the swarm and I show the dramatic consequences of very simple attacks. This is followed by a comprehensive theoretical analysis of the problem to identify potential solutions.
I study possible applications of blockchain technology to secure inter-robot communications and the back-end information management of Swarm-SLAM [28]; taking inspiration from the recent impressive results of this technology achieved in different areas, especially in swarm robotics.
Furthermore, I derive a Byzantine fault-tolerant Swarm-SLAM protocol and I bring a custom blockchain on ROS2. In particular, to the best of my knowledge it is the first blockchain-secured SLAM framework. My approach uses custom decentralised programs executed via blockchain technology (blockchain-based smart contracts) to establish secure swarm coordination mechanisms to identify and exclude Byzantine swarm members.
Subsequently, I determine the required tools for experimentally validating the developed security protocol and I extensively analyse the simulation results obtained.

1.5. Scientific contribution

In my thesis, I study and summarise which can be the limits, in terms of robustness, of the most advanced frameworks for C-SLAM from a theoretical point of view. I analyse the consequences of a security fault in such a system, discovering that introducing Byzantine robots in the swarm compromises the entire system and the reliability of the final map. Therefore, I propose a solution to improve Byzantine fault-tolerance of systems that perform SLAM with robot swarms.
In this study, I make the following scientific contributions:

- **Contribution 1:** I evaluate the security properties of a C-SLAM architecture, identifying robustness problems when one or multiple Byzantine robots are present. In particular, I show through simulations the consequences of different Byzantine behaviours on a state-of-the-art framework [28].
- **Contribution 2:** I integrate a custom blockchain (the Toy-Chain [57]) in the Swarm-SLAM architecture based on ROS2 in order to increase the security of the system. For the first time in the literature, I design and test a blockchain-based smart contract able to overcome the limitations due to the presence of Byzantine robots in a collaborative simultaneous localisation and mapping scenario; my protocol has the possibility to be tuned on different security levels based on requirements.
- **Contribution 3:** I perform a computational analysis of the algorithm, evaluating its pros and cons through simulation results in a controlled environment.
- **Contribution 4:** I suggest future research directions based on the results I obtained.

These contributions represent an initial step towards addressing security concerns in highly complex C-SLAM systems. My thesis highlights the significance of security in a world where autonomous robots are expected to collaborate extensively performing self-localisation and exploration without the presence of a central authority. The provision of security guarantees becomes imperative in such a context.

2 | Swarm-SLAM

In my research, aimed at improving security in SLAM with robot swarms, I focus on analysing vulnerabilities arising from incorrect data sharing among robots. My method involves the posterior validation of the publicly disclosed information from the robots. I simulated the creation of inter-robot loop closures—they correspond to the spatial transformation calculated from the geometric verification between two perceptual moments—and developed a blockchain-based smart contract. This digital contract can be directly integrated within a real-world application to mitigate the injection of incorrect loop closures in the map aggregation phase.

In my thesis, I focus on one of the most recent C-SLAM solutions, specifically a framework that, as of the time of writing, stands out as the only one meeting all five key properties of an ideal Collaborative-SLAM system: full-sensor availability, decentralisation, robustness, sporadic operation, and sparsity. This system is known as the Swarm-SLAM framework [28]. This categorisation is explained in Table 1.1.

2.1. Framework

The Swarm-SLAM framework possesses some precise functionalities that are briefly described here.

Neighbour management (decentralisation and scalability):
The neighbour management module continuously tracks reliable communication neighbours by monitoring heartbeat messages. It operates without a predetermined number of robots and adapts to random rendezvous, making it resilient to disconnections. Communication and computation budgets are customised based on individual robot capabilities, ensuring efficient resource utilisation.

Front end (flexibility and sensor independence):
The front end module processes odometry estimates, obtained through various techniques and synchronised sensor data. It extracts both local and global descriptors, supporting a wide range of sensors. The system is independent of the odometry source, enhancing

adaptability and ease of integration. I simulated the fact that, once a scene is registered a unique global descriptor is generated for place recognition among robots and it is automatically associated to an odometry keyframe.

Inter-robot loop closure detection (indirect):
The indirect inter-robot loop closure detection process leverages inter-robot communication to search for loop closures within robot maps, even in the absence of direct observations. It is activated when other robots are within communication range. While communication and network connectivity can be sporadic, the system takes advantage of any available opportunity to enhance mapping accuracy. Once two robots meet, if they have similar enough descriptors one of the two calculates the geometric transformation (e.g., the loop closure) between the two images associated with the descriptors. This process implies the exchange of the images. The authors of [28] emphasise the importance of minimal information exchange to avoid bandwidth saturation, and only one of the two robots in the rendezvous receives the other's image to make calculation and avoid unnecessary exchanges.

Back end (decentralisation and optimisation):
In the back end, intra-robot and inter-robot loop closure measurements, along with odometry data, contribute to the creation of a pose graph. Local pose graphs are shared with a selected robot for 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.

2.2. Open problems

In single-robot SLAM, the robot simultaneously estimates its location and constructs a map of its surroundings. The main issue is that the error in the robot's position can accumulate over time, leading to inaccurate maps. A loop closure, in this context, refers to the process of detecting and incorporating information about the robot's revisit of a previously visited location within its environment. Intra-robot loop closures are fundamental for improving the accuracy and consistency of a SLAM outcome.

In a C-SLAM scenario, things are more complex due to the presence of multiple robots that are collectively exchanging information with the goal of accelerating convergence towards a unified map thanks to collaboration. Specifically, they are able to create inter-robot loop closures, that are a multi-agent extension of the intra-robot case, while the latter is still present. Clearly, inter-robot loop closures require information sharing among participants, and their creation needs that the same location is visited by multiple robots. Consequently, security vulnerabilities that permit information sharing from robots with faulty or malicious behaviours can pose significant risks. For instance, when robots are allowed to freely participate in a collective action and one of them behaves in a way that is not beneficial for the swarm, this leads to overall performance degradation and the swarm needs a mechanism to avoid it [52] [23].

Moreover, security must be guaranteed in decentralised C-SLAM systems where the single point of failure of a central server that aggregates the information coming from the swarm components is removed. Decentralised systems face constraints due to the computational and communication abilities of the robots they rely on. To achieve precise simultaneous localisation and mapping estimates, these systems need advanced data management and record-keeping techniques [31]. As a consequence the front end and back end, the typical components of a SLAM architecture, are different in functionalities with respect to the centralised case. In this discussion I focus on the back-end submodule, since it is the part of the system in charge of aggregating shared information and it contains lots of security vulnerabilities. More precisely, a final outcome of a SLAM algorithm is what is called the pose graph: a mathematical representation of the relationships and relative positions of the various poses of the robots in the environment. Indeed, the back end is where pose-graph optimisation take place, this task is performed by a robot elected following some mechanism. In other words, pose-graph optimisation is the procedure that aims to produce the most reliable map based on updated noisy measurements acquired by the robots. A new pose graph is generated and shared among the swarm.
The control of the information flow that is managed by the back end is crucial, since once data are registered in the pose graph, removing them is very difficult, or even impossible, as no method to do so has been proposed. Hence, it is important to protect the pose graph from the injection of unreliable data.

Although state-of-the-art C-SLAM methods are called “robust”, authors often mean robustness against outliers due to perceptual aliasing [59]. They define an outlier as data that deviates significantly within a set of measures, but if the measures are sparse and very different in magnitude from each other they are hardly identified and rejected. Solvers based on graduated non-convexity—a popular technique for the optimisation of a generic

non-convex cost function recently used in PGO [59]—are very efficient for outliers rejection in spatial perception. However, these methods applied in C-SLAM remain very application-specific, mainly related to the computer vision problem of place recognition. I demonstrate that they cannot cope with big outliers, or even with groups of wrong loop closures injected by Byzantine robots. They are accepted as correct measures.
However, robustness in robotics refers to the ability of robotic systems to continue functioning and maintaining their properties even in the presence of unexpected events or perturbations [43]. These events can occur for various reasons and pose a significant challenge in the development of autonomous artificial systems. Reliability, instead, is the ability of a system to consistently perform a specific function under certain conditions without failure. These features are strictly related to each other and they are highly desirable in artificial systems, allowing machines to confront new challenges and to work autonomously.

When it comes to SLAM applications, reliability and robustness of the map become even more critical than in other contexts. This is because a large part of the operations performed by the robots rely on their ability to know where they are and efficiently move in the environment based on the SLAM procedure outcome. Recently, it has been observed significant advancement in the field in terms of performance, such as accuracy and speed of SLAM systems. However, real-world implementations are still limited due to the broad range of disturbances these architectures may encounter [4]. The authors of [4] note that perturbations in such complex systems can stem from diverse sources and have a substantial impact on the final result, particularly in long-term operations.

In a typical SLAM scenario, the major sources of errors can be categorised as follows [1]:

- Errors caused by the **sensors**: occur when a robot cannot process collected data accurately, or when noise is present in the data during collection.
- Errors caused by the **environment**: involve sudden performance degradation due to changes in the environmental conditions.

Addressing the errors originating from these sources is a crucial research goal. As previously discussed, robot swarms can help mitigate some of these errors due to the redundancy (of sensors and robots) and due to their high numbers and ability to cover the environment. However, deployment SLAM algorithms in robot swarms is difficult due to the complexity and challenges of controlling decentralised systems.
Kegeleirs et al. [26] explain these challenges as follows.
Individual robots are viewed as components within a larger system comprising multiple entities. While individual task performance remains important, a global map emerges

through swarm collaboration. Moreover, the natural redundancy coming from swarms of robots helps addressing problems when robot's components or sensors become unreliable, avoiding operational failures.
So, in the context of C-SLAM, each robot is considered a component in the global swarm and the final goal should be to have a fault-tolerant overall system. This leads me to identify individuals as possible sources of error that impact the final map.

While C-SLAM research has made significant progress in enhancing sensory and environmental robustness [5, 26], focusing on aspects such as improving map quality by introducing the possibility to scale the number of robots [28], managing information exchange, and sensor fusion, there is a notable absence of studies or implementations related to fault tolerance in SLAM with robot swarms. Current C-SLAM systems operate as expected, creating shared maps without a global reference frame and minimising perception errors, only in the case of all robots adhering to some form of ideal behaviour.

Particularly, by studying the implications of the presence of Byzantine robots within the swarm, I aim to eliminate the assumption that every robot behaves in a predefined manner during collaboration. In the following sections, I provide a comprehensive evaluation of this extension, highlighting the necessity for secured systems.

2.2.1. Threats of incorrect information injection

Protecting the system from the injection of incorrect loop closures by Byzantine robots is of central importance, since loop closures are fundamental for pose graph optimisation. Therefore, I want to ensure that Swarm-SLAM is secure against unauthorised inter-robot loop closures and Sybil attacks [13], i.e., a type of security threat in which a single adversary, by creating multiple fake identities, gains control over a large fraction of the network. I tackle this problem in the Section 3.3.

In Swarm-SLAM, a loop closure is generated when a robot (let's call it Robot i) recognises the same scene that another robot (Robot j) saw in the past. When they meet, Robot j sends its image frame relative to the corresponding scene to Robot i, and Robot i calculates the geometric loop closure. I found that a loop closure can have two main sources of error:

1. Wrong loop closure calculation from Robot i.
2. False information sent by Robot j or alteration of the information from the one that is receiving the data.

I describe issue number 1 here and issue number 2 in Section 2.2.2, since a major challenge in C-SLAM in general arises when the Byzantine robot is the one sending the information.

The latter case presents a wider problem of information reliability in sparse systems, and it is intrinsically not solvable with direct methods, nor addressed yet in other ways.

In Swarm-SLAM, the exchange of information must be minimal. During rendezvous, only one robot sends its useful data to the other, and not vice-versa. The receiver calculates the geometric transformation and only the loop closure, a sort of high-level data, is used for PGO. If each robot sent its images to every encountered peer about the individuals encountered in the past for direct verification, the system would not be scalable. Instead, by collecting only the loop closures on the blockchain, we can save and exchange lightweight yet fully informative information in a secure way. Each robot keeps and exchanges its own perception data only.

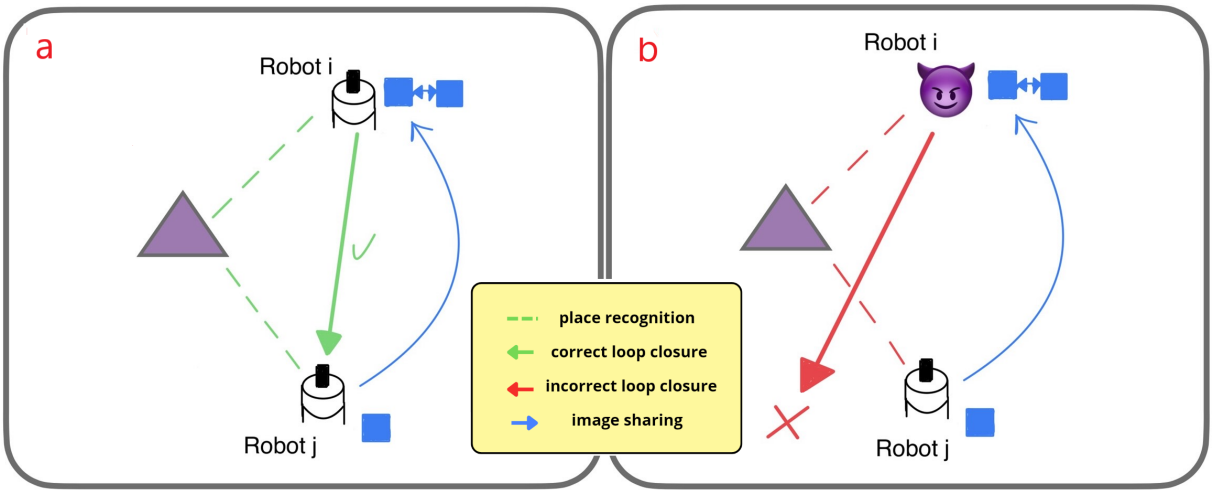

Figure 2.1: The dashed red lines indicate that, in different moments, two robots acquired information about the same scene (illustrated through a purple triangle); which corresponds to two matching descriptors during the meeting. In this scenario, Robot j sends the image (blue square) that has the matching descriptor to Robot i.
In panel **a** is depicted the correct process that should take place in case of a rendezvous between two reliable robots. The green arrow represents a correct inter-robot loop closure from Robot i to Robot j.
Contrarily, in panel **b** is represented the construction of an incorrect loop closure. Robot i is a Byzantine robot (represented by the purple evil) and it generates an incorrect loop closure (red arrow) from Robot i to Robot j.

Currently, every agent could theoretically calculate false geometric transformations concerning the perceived data. Hence, they could create loop closures that work as wrong constraints in the pose graph optimisation, leading to incorrect results. Figure 2.1 illustrates a comparison between a correct process, where an exact inter-robot loop closure is

calculated from Robot i to Robot j, and another scenario in which the sender of the loop closure is a Byzantine robot, resulting in a wrong outcome.

2.2.2. The information reliability problem in sparse systems

In this subsection, I describe the second fundamental security problem in Swarm-SLAM, i.e., information reliability. In particular, information reliability seems to be a general issue related to sparse systems (e.g., robot swarms) where single entities collaborate to achieve a common goal, relying on asynchronous and non-verifiable information retrieval. In these cases robots share data collected locally, both in space and time, and no means of verification is always possible from the rest of the swarm. This open problem can be easily extended to a variety of other engineered swarm systems where the hypothesis of reliable or honest robots is removed.

Addressing this point in Swarm-SLAM is intricate, particularly because it pertains to past events when the Byzantine robot collected potentially false perceptions, that are exchanged in future moments. It can be seen as a general case that involves identifying the robot with faulty visual perception, the Byzantine-malicious robot that sends false images, or even the malicious robot that lies about the association between correct images and odometry steps. Some aspects of this problem are inherently solved by the Swarm-SLAM without additional security mechanisms, as a Byzantine robot creating a random descriptor-image couple and finding a match with real descriptors has a very low probability of happening. However, the malicious or faulty robot might share slightly modified images, introducing noise that leads to the calculation of wrong loop closures. This situation is in some way similar to a more general bad image-odometry association.

Despite the consequences of a fault related to inaccurate descriptor-odometry matching primarily affecting the collocation of the individual maps into the graph, I define two distinct objectives to ensure that the PGO is not compromised by incorrect links:

1. **Confirmed matching between descriptors and odometry in keyframes:** Descriptor-odometry correspondence refers to the need for a correct association between the compact descriptor, that directly refers to a scene, with an odometry pose at every increment of the robot's trajectory. Preventing Byzantine robots from changing the positions (the nodes of the pose graph) at which constraints are referring to (the loop closures) is essential to obtain coherent maps, avoiding instability of results and ensuring convergence of the pose graph. Figure 2.2 shows the difference between a correct descriptor-odometry association along a trajectory and an incorrect one, depicting the fact that in the second scenario a Byzantine

robot modifies the sequence of data that it recorded.

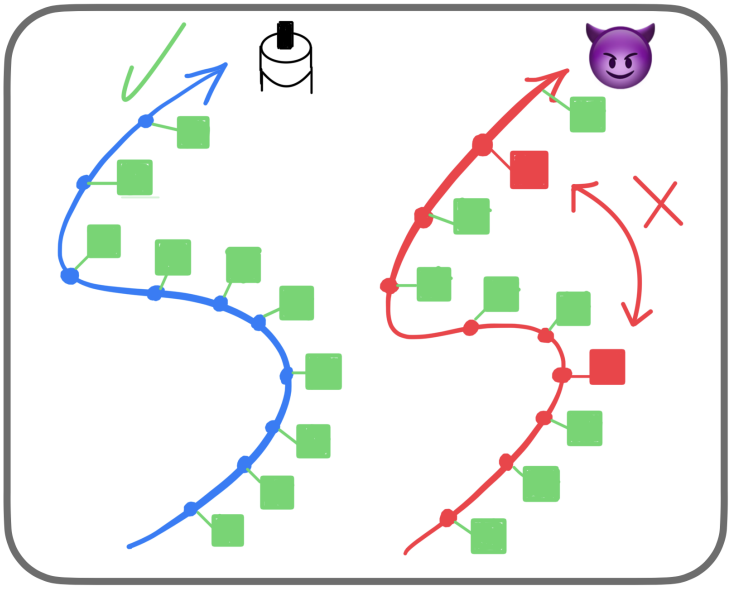

Figure 2.2: Illustration of a descriptor-odometry mismatch. The blue trajectory on the left is the path of a honest and reliable robot, that correctly associates the images it acquires with the correspondent odometry measure (blue points) for each keyframe (green squares). In the red trajectory on the right, a Byzantine attack is represented, two images (red squared) are exchanged with respect to their keyframes. The descriptors are real ones, so some other robots could match them and perform wrong calculations during future rendezvous.

Blockchain can effectively enhance security in the architecture, improving the reliability of matches between descriptors and odometry in keyframes, thanks to its tamper-resistance and immutability. Currently, every agent could theoretically share different data concerning the perceived ones, and they could create loop closures using images perceived during past keyframes, e.g., odometry steps. Registering every keyframe data on a decentralised database (e.g., a blockchain) is memory intensive and not practically realisable, even potentially useless, since in the Swarm-SLAM framework trajectories are considered noisy objects that have to be corrected. I propose that every robot should be free to collect the data it wants, but it is obligated to locally sign the image-odometry couples only in the keyframe associated to a candidate loop closure. In this way I force Byzantine robots to register their data in an immutable manner on a decentralised database, accessible from everyone. This approach, allows the recording of the descriptor-odometry couples in time, enabling other robots to verify the sequence and associations before computing the

loop closures. On the blockchain, the image's descriptor is registered together with the information related to the odometry within a transaction.

2. **Ensuring perception data reliability:**
   Once we can trust that the proposed descriptor-odometry pairs are registered and verifiable, the next step is to check the quality of the data. Images, or even LiDAR point clouds registration is associated with a global descriptor for every keyframe. Descriptors must be saved on the blockchain at the moment of the loop closure recognition. Hence, every robot of the swarm is in principle able to check that the image received is not altered, recalculating back its descriptor and comparing it with the one in the database. This ensures the complete reliability of the data used in loop closure generation.

   Figure 2.3 depicts the consequences of bad data shared from a Byzantine robot to a second honest robot, making a comparison between a scenario in which everything goes smoothly and one in which an incorrect inter-robot loop closure is generated.

   My theoretical analysis highlights how, without any security protocol, the system is vulnerable to false image injection: if a Byzantine shares a bad image during the rendezvous with the other robot that is calculating the geometric transformation, the loop closure results are wrong. The honest robot never realises that it has damaged the system.

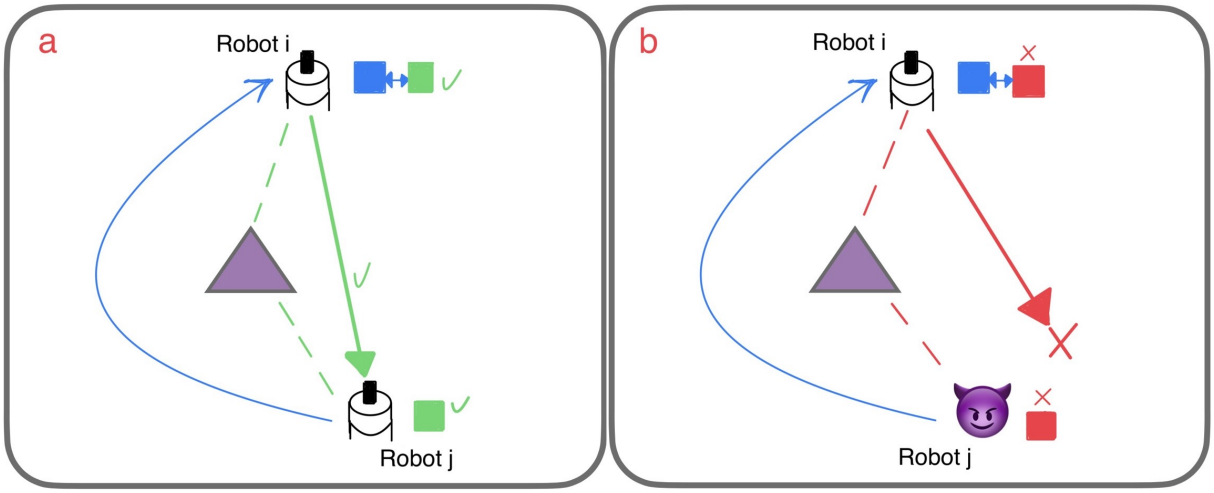

Figure 2.3: The dashed red lines indicate that, in different moments, two robots acquired information about the same scene (illustrated through a purple triangle); which corresponds to two matching descriptors during the meeting.
In panel **a** is depicted the correct process that should take place when two reliable robots construct an inter-robot loop closure during a rendezvous. The green arrow represents a correct inter-robot loop closure that is generated from Robot i to Robot j using the correct image (illustrated as a blue square) that Robot j shares with Robot i.
Contrarily, in panel **b** is represented the construction of an incorrect loop closure. In this scenario, the Byzantine robot (illustrated as a purple devil) shares a wrong perception data (e.g., an image illustrated as a red square indicating that is a perturbed information) with the honest robot that, as a consequence, calculates a wrong loop closure.

2.2.3. Need for fault-tolerant, robust, and secure systems

Preventing collaborative systems from false data injection and mitigating malicious attacks is a necessary step towards robotics applications with safety guarantees [23].
In this chapter I presented the principles of the Swarm-SLAM framework and elaborated a comprehensive theoretical analysis on the main security challenges that seem not to be addressed yet in the C-SLAM literature. I categorised the issues that can be solved independently and subdivided them in smaller problems. I performed an analysis of interesting possible solutions that can be realised by leveraging blockchain technology.

In Swarm-SLAM, even the injection of a single incorrect loop closure in the pose graph is hazardous for the map generation and the simultaneous localisation of every robot. One incorrect loop closure can lead to a very high localisation and mapping inaccuracy.
The ideal example in which only one incorrect inter-robot loop closure exists is intentionally oversimplified and very far from reality, but it gives the motivation for the rest of

the work. In a realistic scenario with lifelong experiments hundreds of loop closures can be proposed by a single robot. It is straightforward, to deduct that a Byzantine robot can pose a significant risk to the entire system, with catastrophic consequences in safety-critical missions.
In the representation of the swarm's final pose graph on a map, accuracy is expected when compared to the ground truth of the global map. Specifically, even in scenarios where robots exhibit no noise in the odometry data, implying perfect localisation, and the pose graph optimisation only consolidates the trajectories of eight robots in a global map through inter-robot loop closures generation, the presence of some incorrect loop closures consistently compromises the optimisation.

Here, from a more general perspective and without limiting to any specific case from the categories I defined previously, I want to present the consequences of having one single Byzantine robot that for some reason is able to negatively influence its loop closures. Swarm-SLAM cannot recognise incorrect transformations or stop them.

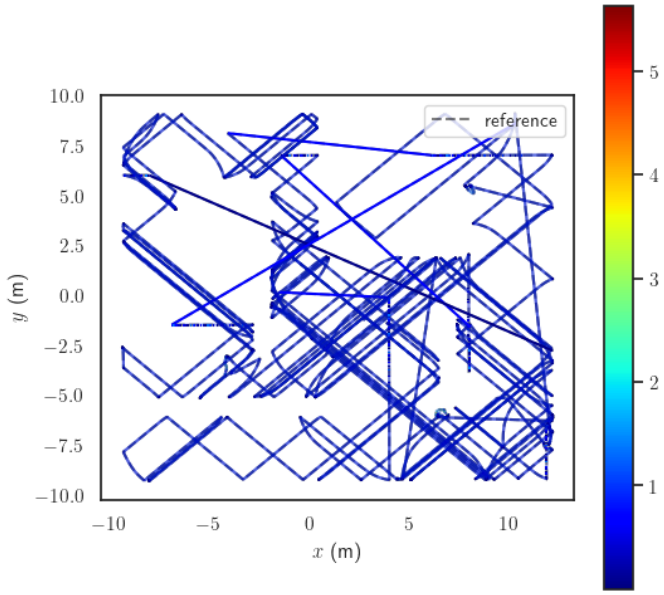

Figure 2.4: The pose graph composed by the trajectories of 8 reliable robots (coloured line) is compared with the ground truth (dashed line). The colour bar on the right indicates the amount of error at each point of the pose graph. A blue pose graph indicates that the absolute positional error is in general very low (the dashed line is overlaid by the coloured one). The trajectories of the pose graph are aligned with the ground truth using Umeyama algorithm [19] as described in Section 4.2.

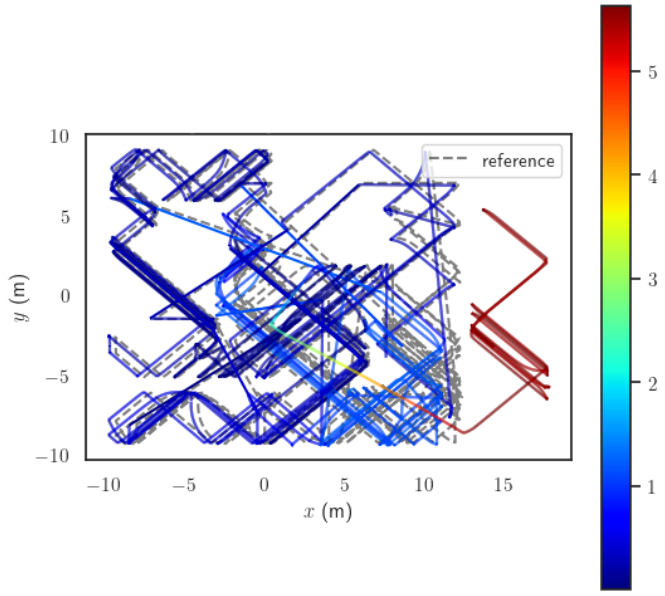

Figure 2.5: The pose graph (coloured line) composed by the trajectories of 7 reliable robots and 1 Byzantine robot is compared with the ground truth (dashed line). The colour bar on the right indicates the amount of error at each point of the pose graph. The overall pose graph is consistently different from the ground truth and the portion of the graph in red corresponds to very high absolute positional error. The trajectories of the pose graph are aligned with the ground truth using Umeyama algorithm [19] as described in Section 4.2.

From the comparison of the pose graphs shown in Figures 2.4 and 2.5, it is evident the negative effect of wrong loop closures. Not only the trajectory and map of the Byzantine robot are completely distorted; but also the global aggregated map contains large errors with respect to the ground truth. During the pose graph optimisation procedure, the trajectories are modified to respect the imposed constraints, if these constraints are wrong the good trajectories are affected by the changes. To highlight these negative effects, these simulation results refer to perfect trajectories with no noise in the odometry of any robot. To evaluate and quantify the error that is generated on the overall map due to a Byzantine robot, statistics on the Absolute Positional Error (APE) are reported in Figures 2.6 and 2.7, i.e., the difference at every keyframe between the actual and the estimated map. What gives an indication of how the general map is perturbed by the wrong transformation is the Root Mean Square Error (RMSE) over the APE distribution, that is more than 1.5 metres of average error in one case and almost zero in the other. Peaks of several metres of error in a map that cover an area of about 100 square metres correspond to very high inaccuracy.

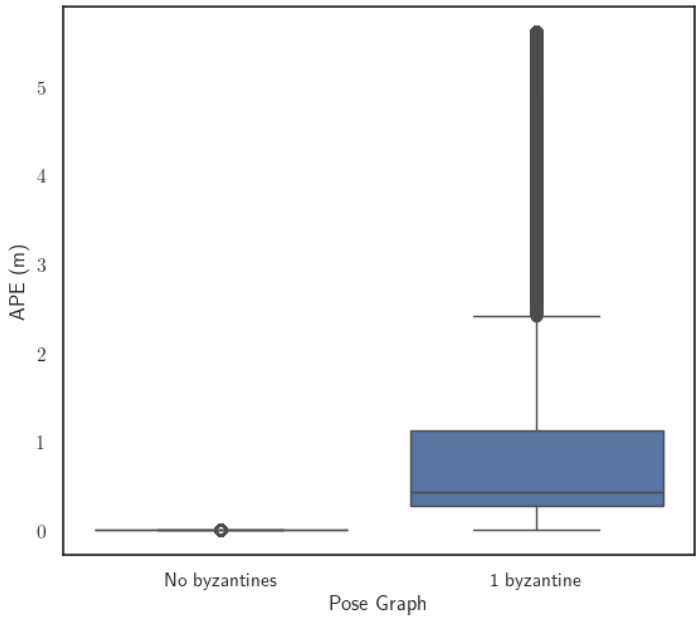

Figure 2.6: Absolute Positional Error (APE) distribution. Further details about this type of plot are discussed in the caption of Figure 4.5.

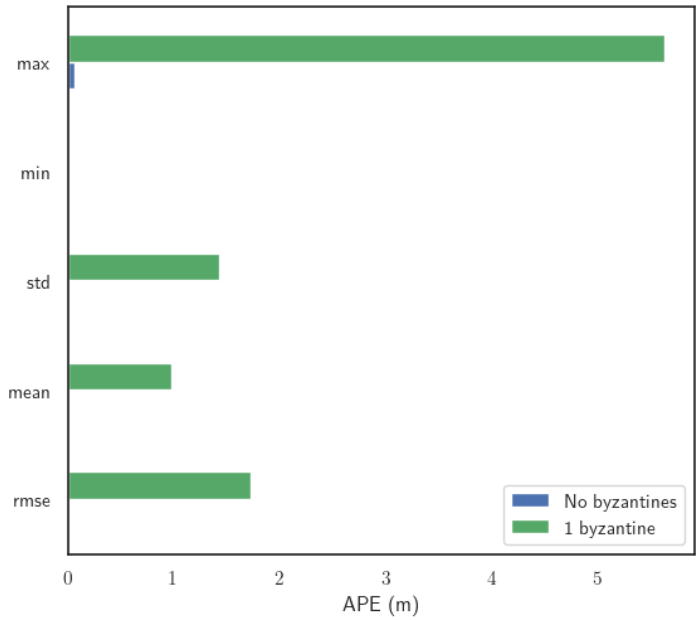

Figure 2.7: APE Error statistics. In this plot are reported the Root Mean Square Error (RMSE), the minimum value (min), the maximum value (max), the mean, and the standard error (std) related to the APE distribution as described in Section 4.2.

These observations bring me to the conclusion that blockchain-based smart contracts may enhance security in C-SLAM in general, especially when it pertains to data verification, reputation management, and task evaluation. Hence, in the next chapter, I develop high-level consensus strategies to verify loop closures' correctness before their usage in the optimisation phase, and I prove that robot reliability can be measured through blockchain transactions. However, ensuring the reliability of perception data remains a minor aspect of my contribution. It is strongly application-dependent and advanced metrics can be studied in future works that are not discussed here, to provide an indication of the data quality looking at the overall map composition.

However, it is also worth saying that the theoretical considerations I have presented so far are applicable to any C-SLAM framework, and the methods based on blockchain that I elaborate upon in Section 3.3 are general and framework-independent, with the potential for adaptation to the working principle of various SLAM systems.

3 | Byzantine fault-tolerant protocol

A Byzantine fault-tolerant protocol is a class of methods designed to handle failures, including malicious failures or arbitrary faults, in a distributed system.
In a Byzantine fault-tolerant system, the working principles are designed to tolerate erroneous and potentially malicious behaviours from a certain percentage of nodes or components within the system, included entities that may send incorrect information intentionally.
The building blocks of a Byzantine fault-tolerant system are [32]:

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

To provide high levels of resilience against arbitrary and malicious faults, ensuring the reliable functioning of a Swarm-SLAM system, I decided to leverage blockchain technology.

3.1. Why blockchain?

As already introduced, the term "Byzantine" derives from the Byzantine generals' problem [32], a conceptual problem in distributed computing that involves coordinating a group of agents that may not all be trustworthy.
In the context of Swarm-SLAM, Byzantine may refers to robots that in some way can lead to a bad contribution in the map optimisation, intentionally or not. Since odometry is already considered a noisy quantity in the Swarm-SLAM pose graph optimisation process, in this discussion odometry noise is not examined as a cause of Byzantine behaviour. Byzantine robots are robots with a behaviour that negatively influences the creation of loop closures, which are the hard constraints of the optimisation problem.

The advancements in cryptographically secured distributed ledgers, such as blockchains, permit the use of decentralised computing platforms to control the injection of information in systems where direct verification among participants is not possible, preserving privacy and scalability [44]. Moreover, blockchains can be used to manage robots' permissions inside a network, changing the robots' decision power in the swarm based on reputation [52] [54].

When facing the challenge of controlling loop closures injection into Swarm-SLAM, smart contracts running on the blockchain have the role of a distributed filter that approves only the information that possesses certain requirements. This is tremendously helpful when the goal is to protect the PGO from incorrect information because the map is never directly affected by Byzantine attacks.
Compared with a centralised controller, blockchain gives several advantages. Thanks to its decentralisation, it can be completely integrated within Swarm-SLAM. The single point of failure is removed. Indeed, blockchain distributes the control of information among the swarm, reducing the risk of having only one central server which would not be desirable in Swarm-SLAM. A blockchain also ensures that the data it stores is tamper-proof and thus cannot be changed by an ill-intent robot. Lastly, the possibility of having identical smart contracts that are guaranteed to be executed in the same way by everyone ensures trust among system users.
These are just few key factors that motivate me to propose the use of blockchain to improve the reliability of Swarm-SLAM.

I build my idea by taking inspiration from the promising results achieved by Strobel et al. in their work on how to use blockchain to secure robot swarms [53], that confirmed to be a successful approach to protect a robot swarm from Byzantine faults. However, as Reina highlights in his theoretical analysis on how to secure robot teams with blockchain

[45], some aspects must be carefully considered when blockchain is used on robots. For example, the choice of the consensus protocol.

In my implementation, I use the Toy-Chain's smart contracts to filter bad information originated from the robots. Every robot is a Toy-Chain node and all together they are able to maintain the blockchain network with simulated peer-to-peer communications. As consensus protocol I use Proof of Authority (PoA)—the standard protocol implemented on the Toy-Chain—that means that robots who can validate transactions are elected based on their reputation in the network. On the contrary, in Proof-of-Work (PoW)—the protocol implemented in Bitcoin [38]—the consensus relies on computational power, but PoW can be infeasible in robot swarms, due to the limited computational capabilities of the robots.
In Swarm-SLAM computational resources are already heavily used in data processing and optimisation, therefore the choice of PoA is the most logical one.

When Swarm-SLAM is secured by the Toy-Chain, robots that want to propose an inter-robot loop closure for the PGO, cannot do it directly because the smart contract has to first run a filtering mechanism. A loop closure can go into the pose graph optimisation phase only after that the smart contract runs some checks to select or reject loop closures. Once the consensus is achieved and the execution of the smart contract leads to the approval of a new set of loop closures, these are accepted by the robot in charge of performing the optimisation of the pose graph. Moreover, in this way I force robots to register their data in an immutable manner on a decentralised database, transparent to everyone who wants to check the information, thus making possible to identify Byzantine robots. Figure 3.1 depicts the workflow of the Toy-Chain when a transaction is registered from a robot.

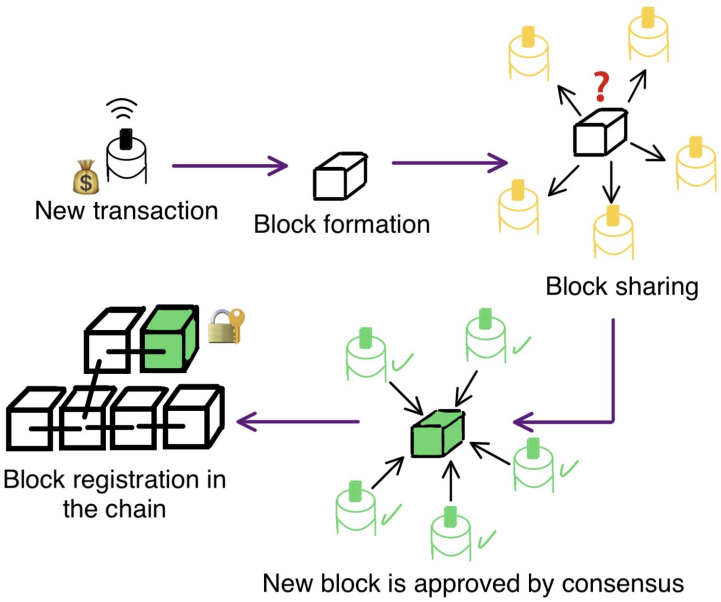

Figure 3.1: Workflow of the Toy-Chain when a transaction is registered from a robot. In step 'New transaction' it is represented a robot that wants to propose a new transaction, that in my case corresponds to the registration of an inter-robot loop closure. The robot must stake some amount of digital 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 happen 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.

3.2. Design

I designed and implemented a blockchain-based smart contract, i.e., an algorithm running on the blockchain, that is able to check security requirements over the loop closures that are proposed by every robot in order to prevent the injection of permission-less information in the Swarm-SLAM back-end. This smart contract is based on geometric relationships that the correct transformations between robot poses have. It is able to reject incorrect loop closures, at the cost of introducing latency in the system.
Once a set of inter-robot loop closures are validated from the blockchain, they are auto-

matically shared among the robots from the one that measured them, and they can be used to improve the map's accuracy. 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. Such raw heavy data do not need to be further considered once the relative loop closures are created and incorporated into blockchain transactions, preserving scalability in swarm systems.

A blockchain transaction, in my case, is an action or an operation that a robot initiate on the Toy-Chain network. Every Toy-Chain transaction has a sender and it is linked to a function call on the smart contract that triggers a specific operation. When the transaction is processed, it can lead to changes in the state of the Toy-Chain's smart contract, based on the algorithm that I implemented in order to verify loop closures. The robots send a blockchain transaction every time they want to propose a new inter-robot loop closure.
My method can be seen as a 'reputation management' system where robots stake their reputation in the form of a digital asset, and they increase it or lose it depending on their actions. In particular, a robot needs some reputation to make transaction-secured actions. I use stacking protocols where each robot possesses a finite amount of wealth, and it has to pay a fixed quantity of it to accomplish actions on the Toy-Chain. The effect of this is to limit the ability of Byzantine robots to harm the system [54].

In practice, robots that encounter each other and have matching descriptors can only propose a new inter-robot loop closure when they have enough crypto tokens to stake. Only one of the two can propose it, based on the outcome of the smart contract that decides who the sender and the receiver are. A crypto token is the digital asset that represents a unit of value within a blockchain-based system.
The elected robot stakes some crypto tokens to register a new transaction, which is then added to a block and propagated throughout the swarm as robots encounter each other.

A transaction registers and makes public the descriptor of the scene, the transformation, the ID of the sender, the ID of the receiver and the poses with the corresponding keyframes, among other information (see Table 3.1). The registration of the descriptor, along with the information related to the odometry, firmly establishes the descriptor-odometry correspondence at the keyframes where inter-robot loop closures are calculated, thereby mitigating the impact of Byzantine robots. Hence, when a robot proposes a loop closure, every other component of the swarm is subsequently able to verify the data associated to it, preventing the possibility for a Byzantine robot to manipulate them. Indeed, this capability enables the swarm to perform posterior information verification and met-

rics calculation on consistent data, establishing a foundation for addressing the security problems outlined in Section 2.2.3.
In this section, I refer to the concept of information blocking in sparse systems, enhanced by blockchain technology, as a foundational building block to obtain a Byzantine fault-tolerant Swarm-SLAM framework.

$Descriptor$	ID_{Sender}	$ID_{Receiver}$	$Pose_{Sender}$
$Keyframe_{Sender}$	$Pose_{Receiver}$	$Keyframe_{Receiver}$	$Transformation$

Table 3.1: This table depicts the information stored in a transaction (Tx). A transaction is composed of eight data fields, here represented on two rows for visualisation simplicity.

3.2.1. A filtering smart contract based on geometric constraints

The major research contribution of my thesis is the development of a blockchain-based smart contract that leverages on geometric constraints to validate information.
I focus my analysis to the case in which the Byzantine robot is the one that generates a wrong loop closure; this is undoubtedly the easiest and the most deleterious action that a malicious agent can perform against the swarm.

In the unprotected Swarm-SLAM, when a generic robot proposes a loop closure, it is directly used for PGO. Indeed, good and bad measures are not distinguished from each other. Since loop closures are generated based on images or other data collected in the past by two individuals in separate moments, there is no way to perform direct checks on their validity.
When solving complex spatial problems, geometry often proved to be the solution. Additionally, temporal dynamics play a crucial role here, and direct verification is impossible. The idea is to consider an abstraction of the problem and looking only at the geometric structure of the interconnection between the nodes of the pose graph, namely, looking at the loop closures as pure geometric transformations between poses in the plane.
As the number of inter-robot loop closures generated increases, numerous triangles whose sides are the single transformations, start to generate among the robots that recognise the same scene and share similar descriptors. I leverage the triangle identity to verify that this spatial constraint is respected, before allowing any robot to insert new information into the system. This mechanism is illustrated in Figure 3.2.
Every time a new transaction occurs on the Toy-Chain, the algorithm executed via the smart contract evaluates potential new triangles that can be generated. Approved loop closures result from candidate loop closures meeting the geometric constraint, and the corresponding senders receive rewards. While the other candidate loop closures remain

in a pending state with no stake returned to the owner.

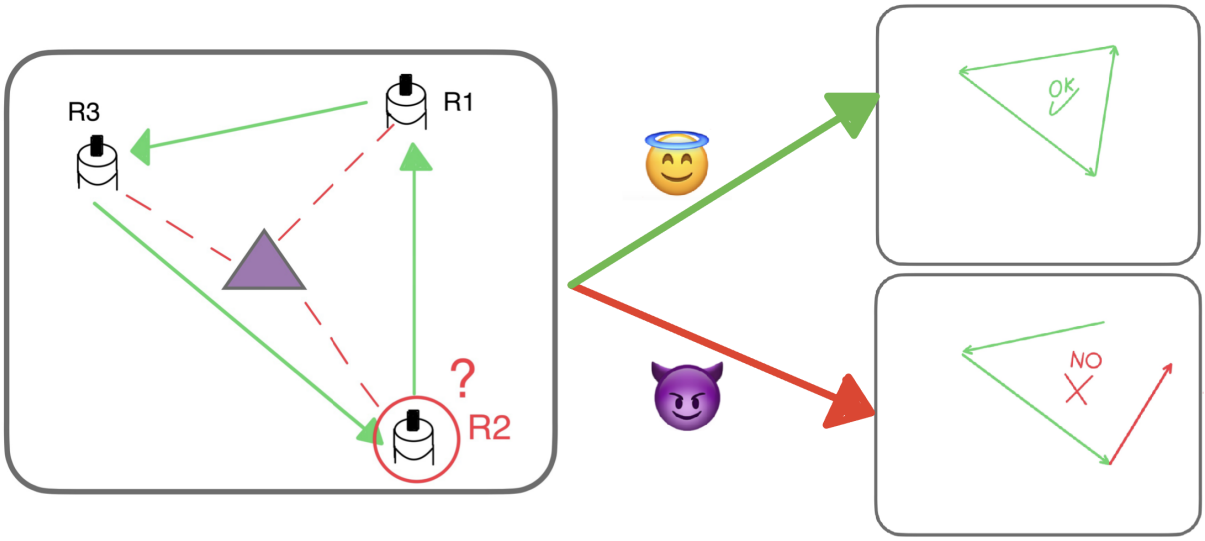

Figure 3.2: This figure depicts the two possible outcomes of a smart contract verification. The green arrow is the case of three honest or reliable robots. The red arrow illustrates an incorrect inter-robot loop closure created by a Byzantine robot, where the triangular identity is not respected. The inter-robot loop closures cannot be accepted until verified and, as a consequence, they remain unconfirmed.

An additional constraint on the object construction is imposed. Since a triangle is formed from three different robots, at least three meetings between these agents must happen before its evaluation. Specifically, to maintain homogeneity in collective roles, every robot is required by the smart contract to generate a loop closure towards the robot that is pointing to the third. It leads to the creation of a cycle in the exchange of heavy raw data, promoting peer-to-peer verification, while banning a Byzantine robot from being successful. At least two Byzantine robots over three must collude to cheat a security check.

A control parameter allows tuning the sensitivity of the smart contract against noise, when computing the transformation, i.e., the loop closure. It represents the limit on the Euclidean distance measured between the head and the tail of two transformations, which, in a perfect triangle, should match at every vertex. In the ideal case of noiseless loop closures, the control parameter is zero, indicating a perfect match for the triangle. In practice, some error is always introduced in the inter-robot loop closure calculation due to imperfect geometric verification, or due to Byzantine behaviour. It is possible to reject loop closures over a specific threshold of error, setting the control parameter.

Another feature of the smart contract I propose is the possibility to force the robots to generate candidate loop closures in predetermined ways, otherwise their candidates are discarded. In the non-secured system where no triangles are required, the sender of the inter-robot loop closure is randomly selected during each meeting. Conversely, in a blockchain-secured system, the smart contract compels pairs to choose the sender in a manner that maximises the number of completed triangles. It is essential that for a set of arrows to form a triangle, their orientation must follow a cyclic pattern respecting the head-tail correspondence. Robots are forced by the smart contract to follow this rule.

My Byzantine fault-tolerant framework is also able to automatically preserve the map accuracy from other unreliable sources of Byzantine information. More precisely, Byzantine's trajectories are never included in the pose graph. In Swarm-SLAM, a trajectory must be linked through loop closures within the global map, in order to be part of the global pose graph. Hence, the local map of Byzantine robots is added to the global map only when it succeeds in creating a loop closure with a well-functioning robot. Thanks to my protection mechanism based on triangle verification, Byzantine robot's loop closure are never accepted and therefore Byzantine trajectories are never merged in the global map.

3.2.2. Security level parameter

The security level is an additional parameter of a loop closure with two functionalities, it indicates how much an inter-robot loop closure has been validated, and hence, how much it is secure. Consequently, the security level is used to reward robots in a proper way, based on their honest contribution to the map improvement. Precisely, robots receive a reward proportional to the security level of the validated loop closure.
When a transaction is initiated, the security level of the corresponding candidate loop closure is zero. Indeed, no one validated it through a triangle identity verification and no information is approved. While the number of triangles increases, the security level of every loop closure that is participating in a new triangle validation is increased by one. Hence, every approved loop closure will have at least security level equal to one. More precisely, this security parameter corresponds to the number of approved triangles in which a loop closure is involved, constructed by different sets of robots.
The security level parameter of the inter-robot loop closure i, sent by Robot r, is $\delta_i{}^r$.

As an example, suppose that Robot 1, Robot 2, and Robot 3 recognise the same scene through similar descriptors and they validate a triangle. Three loop closures are validated, $1 \to 2$, $2 \to 3$, and $3 \to 1$. Then, their security levels correspond to one. Consider now

that a new triangle around the same scene exists between Robot 1, Robot 2, and Robot 4.

Suppose also that their transformations are $1 \to 2$, $2 \to 4$, and $4 \to 1$. Once the blockchains are synchronised, the security levels are all one, except the loop closure from Robot 1 to Robot 2 that has security level equal to 2. This discussion is extended in Figure 3.3.

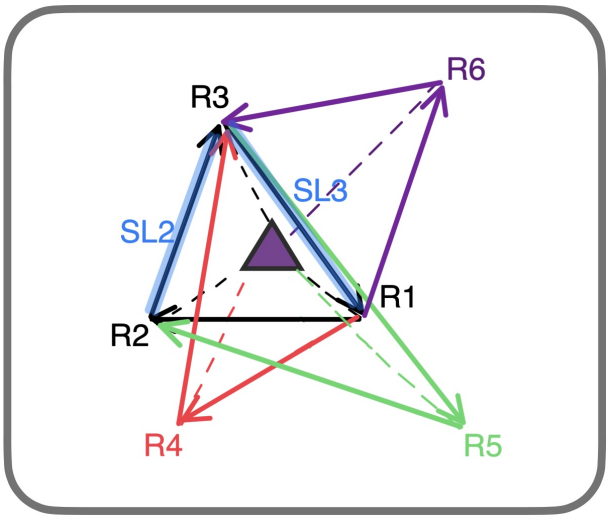

Figure 3.3: Visual representation of the evolving process as multiple triangles are created. The security level (SL) example in the text is extended: six robots (R1, R2, R3, R4, R5, and R6) recognised the same scene (the purple icon), and four triangles (black, red, green, and purple) are measured and 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.

A threshold of acceptance on the security level parameter is encoded in the smart contract that every robot executes. It is possible to vary it, based on the trade-off a user wants to make between efficiency and security. The higher the threshold, the more loop closures are pending until a sufficient number of triangles are generated.
This trade-off has to be carefully evaluated based on the size of the swarm and the requirements for real-time performance. A standard threshold is set to one and it is very effective in case no coalitions among Byzantine robots are present. Higher security levels help to protect the systems against colluding Byzantine robots.

Reputation

In the context of collective robots, reputation refers to the perceived reliability that a robot demonstrated during its social activity. Reputation mechanisms help robots to know if they can trust their peers based on their past contribution to the swarm's goals. Robots with a high reputation have more chances to be trusted, while those with a poor reputation should be treated with caution because they probably injected misleading information. In a Swarm-SLAM application, robots have a high reputation when their loop closures have been used many times to perform triangle verification successfully, while those whose reputation is low and close to zero are very likely to be Byzantine.
Robots' reputation is updated on the Toy-Chain, based on the smart contract outcomes. I assign the reputation to the robots evaluating the number of inter-robot loop closures they propose that are accepted, and on the security level they reached, based on the following relationship:
The reputation of Robot r, after n loop closures validation is: $R_r = \sum_{i=1}^{n} {\delta_i}^r$.

3.3. General architecture

In this thesis, I develop a precise architecture to simulate and test the impact of the designed Toy-Chain's smart contract. My system allows for the controlled creation of loop closures through robots interactions within the arena.
Three application layers can be identified: Simulation, Security, and Optimisation (Swarm-SLAM back-end). The interaction between the three architecture layers is depicted in Figure 3.4.

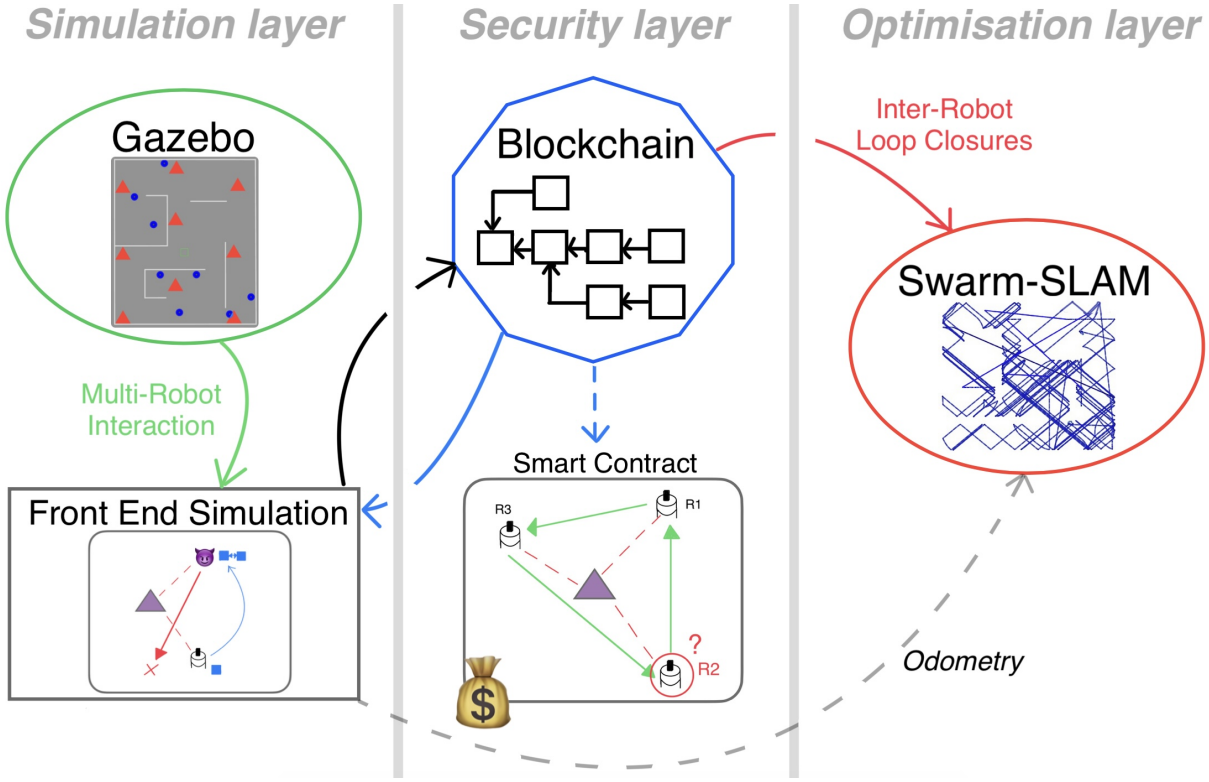

Figure 3.4: The three application layers that compose the system that I used to obtain the results showed in Section 4.3: Simulation layer, Security layer, Optimisation layer.

The simulation layer simulates the environment where the robots move and self-localise. Following the logic described in Section 2.2.3, by moving in the environment, robots generate proposals of inter-loop closures. The simulation layer interfaces with the blockchain. The main contribution of this work is the introduction of the security layer, which includes the blockchain and the smart contract. The security layer interposes between the "physical" source of information (the simulation layer) and Swarm-SLAM (the optimisation layer). The Toy-Chain's smart contract takes the role of a meta-controller and limits robots' individual power.

The optimisation layer corresponds to the Swarm-SLAM back-end, where the information management and the pose graph optimisation take place.

In a real-world experiment things are more complex, since Swarm-SLAM covers much wider roles, from visual scene recognition to map creation. However, through the front-end's rich functionalities and the fact that Swarm-SLAM entirely relies on ROS2, the blockchain should be easily implementable on real robots and I expect to obtain similar results as the one I reported in Section 4.3.

Simulation layer - Noisy odometry model

Odometry is a technique utilised in the realm of robotics and autonomous navigation to measure the changes in both the position and orientation of a mobile robot as it moves in the environment. Typically, this estimation hinges on data collected by sensors installed on the robot's wheels or tracks, facilitating real-time tracking and position updates. These sensors are commonly known as "odometers" or "encoders".
Through the analysis of encoders readings, the robot can compute the distance it has travelled and the extent of its rotation since the last position update.

Odometry plays a pivotal role in numerous SLAM systems. Nonetheless, it has its limitations as it relies on precise measurements of wheel motion, which can be influenced by wheel slippage, uneven terrain, and other variables. Consequently, odometry is often employed in conjunction with other sensor data sources to enhance the accuracy of a robot position estimation.

The first goal of this research is to test the Swarm-SLAM framework and to find its weaknesses in terms of reliability. To do so, I build a simulation environment using Gazebo [27], I reproduce a realistic scenario where robots accumulate noise in their odometry during time. The possibility to fine-tune the noise I want to introduce in my simulated system is of utmost importance, if I want to evaluate the framework's performance. For these reasons, I refer to a widely used probabilistic motion model in order to add noise to Gazebo odometry data.

I briefly explain here the "Sampling-based model for planar motion" [36, 55], and how I use it to introduce customisable odometry error in the simulation output.
The model I am referring to describes how samples of the robot pose increment evolve in presence of a certain quantity of Gaussian, zero-mean random noise. Figure 3.5 depicts a robot's pose increment.

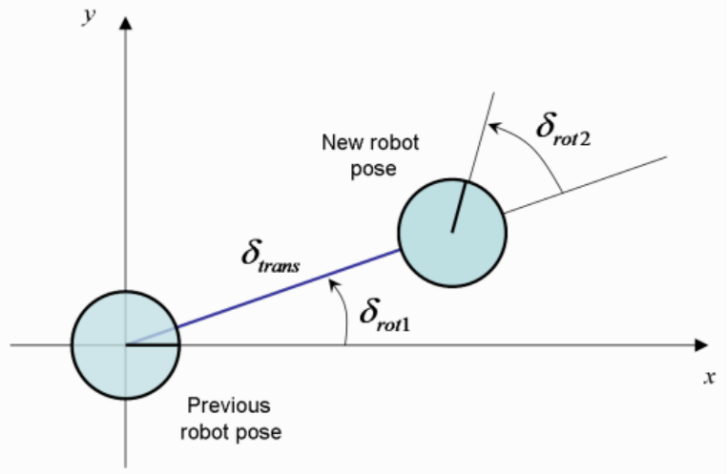

Figure 3.5: A robot's pose increment [36].

As an example, suppose to start from a robot pose that is (0 0 0), the sample of the robot pose increment can be described by the following equation, where (x' y' ϕ') is the new pose:

$$\begin{pmatrix} x' \\ y' \\ \phi' \end{pmatrix} = \begin{pmatrix} cos\hat{\delta}_{rot1} & 0 & 0 \\ sin\hat{\delta}_{rot1} & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} \hat{\delta}_{trans} \\ \hat{\delta}_{rot1} \\ \hat{\delta}_{rot2} \end{pmatrix} \tag{3.1}$$

The variables $\hat{\delta}_{trans}$ (a relative translation), $\hat{\delta}_{rot1}$ (a first rotation at the previous robot pose) and $\hat{\delta}_{rot2}$ (a second rotation at the new pose) are the results of adding the Gaussian noise to the ground truth measures:

$$\begin{aligned} \hat{\delta}_{trans} &= \delta_{trans} + \epsilon_{trans} & \epsilon_{trans} &\sim \mathcal{N}(0, \sigma^2_{trans}) \\ \hat{\delta}_{rot1} &= \delta_{rot1} + \epsilon_{rot1} & \epsilon_{rot1} &\sim \mathcal{N}(0, \sigma^2_{rot1}) \\ \hat{\delta}_{rot2} &= \delta_{rot2} + \epsilon_{rot2} & \epsilon_{rot2} &\sim \mathcal{N}(0, \sigma^2_{rot2}) \end{aligned} \tag{3.2}$$

Where the expressions for the standard deviation approximations are derived in [55] in the form:

$$\begin{aligned} \sigma_{trans} &= \alpha_3 \delta_{trans} + \alpha_4(|\delta_{rot1}| + |\delta_{rot2}|) \\ \sigma_{rot1} &= \alpha_1 |\delta_{rot1}| + \alpha_2 \delta_{trans} \\ \sigma_{rot2} &= \alpha_1 |\delta_{rot2}| + \alpha_2 \delta_{trans} \end{aligned} \tag{3.3}$$

I derive a parametric model with α_1, α_2, α_3 and α_4. They can be tuned based on the

amount of error I want to introduce on the different transformations: degree/degree, degree/metre, metre/metre and metre/degree respectively.
Following a recursive procedure on every odometry increment the noise is accumulated, and as time evolves the error in the pose increases following the noise distribution. Figure 3.6 shows an example where a reference trajectory is compared with a noisy one. I use the same α values reported here for the rest of the work: $\alpha_1 = 150.0$, $\alpha_2 = 160.0$, $\alpha_3 = 1.0$, and $\alpha_4 = 0.001$.

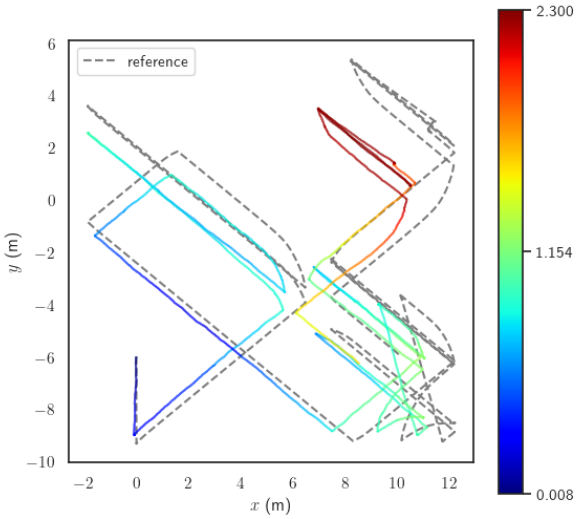

Figure 3.6: In this figure is depicted the trajectory of a robot that is accumulating noise in its odometry (coloured trajectory) with respect to the ground truth (reference). The coloured bar on the right indicates the amount of absolute positional error that is present in every point of the trajectory. The origin of the trajectory is in position $x = 0$, $y = -6$. It can be noticed that, on average, the error is increasing during time.

4 | Simulation results

To test the Swarm-SLAM weaknesses and to examine the effectiveness of my fault-tolerant blockchain-based smart contract, I rely on Gazebo [27], performing extensive simulation experiments. My simulations involve eight robots executing a Swarm-SLAM instance and maintaining their blockchain functionalities. Every robot runs a blockchain node and the connection between nodes is local. Hence, when robots meet, they exchange information calculating loop closures and synchronising the chains.

Focusing on the validation of inter-robot loop closures, which are post-processed high-level information, the Swarm-SLAM front end is not strictly necessary. The 3D map generation is also not an essential element for my tests because in SLAM duality between localisation and mapping is always present, i.e., once localisation is solved, the map can be created and vice-versa. Therefore, I applied simplifications to the Swarm-SLAM simulation that allowed me to include all relevant aspects without losing generality from the theoretical viewpoint.
So, in order to remove unnecessary complexity coming from the front-end in my experiments implementation, I simulated the inter-robot loop closure creation process.
Scenes are simulated as points on the ground and robots perform place recognition by measuring the distance of a pose from the scene, I discuss this process in Section 2.1. Then, the inter-robot loop closure is calculated using geometric coordinates, during rendezvous. Byzantine robots are those that introduce noise in this mathematical operation, modifying the geometric transformation that corresponds to a loop closure.
The number of the scenes can be varied. The Euclidean distance at which a robot recognises a scene and the range of communication between robots can also be modified. These are the free variables of the simulation environment, that can be tuned based on the characteristics of the simulated robots. Although the communication range between peers is the same for both raw data and the blockchain network, in practical applications it can differ due to different communication channels.

All the theoretical motivations remain applicable to this simplified scenario and the results are general. The architecture can be easily applied to a real-world experiment, where the

front end of a complete Swarm-SLAM application is actively used in the loop closures calculation, since the modality of the data generation is the only thing that varies.
The scope of this chapter is to validate the theoretical results, showing the advantage of an additional security layer that aims to limit the actions of Byzantine robots.

I chose to use a total of eight TurtleBots3 [17], model Waffle, as the one showed in Figure 4.1. In my experiments, robots explore the environment depicted in the Figure 4.2, which comprises a number of walls which are treated by the robots as obstacles. The robots perform a random walk, i.e., they move straight until they sense an obstacle and make a rotation in a random direction for a random number of seconds (drawn from a uniform distribution $\mathcal{U}[0.01, 6]$ seconds) at a velocity of 0.5 rad/s. Robots have large noise in the odometry that cannot be corrected without collaborative SLAM as I do not consider in my simulation inter-robot loop closures; therefore, noise in the odometry monotonically increases on average.
When Swarm-SLAM comes into play, robots that previously saw the same scene and now meet, propose to each other potential loop closures between keyframes of their trajectories. A robot in charge of performing PGO begins to generate a global map based on its knowledge. Subsequently, if the inter-robot loop closures are accurate, the localisation of the swarm components is collectively enhanced, leading to a reduction of the accuracy error in the global map.

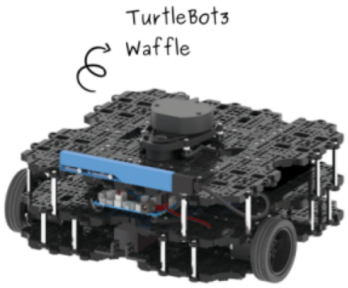

Figure 4.1: TurtleBot3 model Waffle [17].

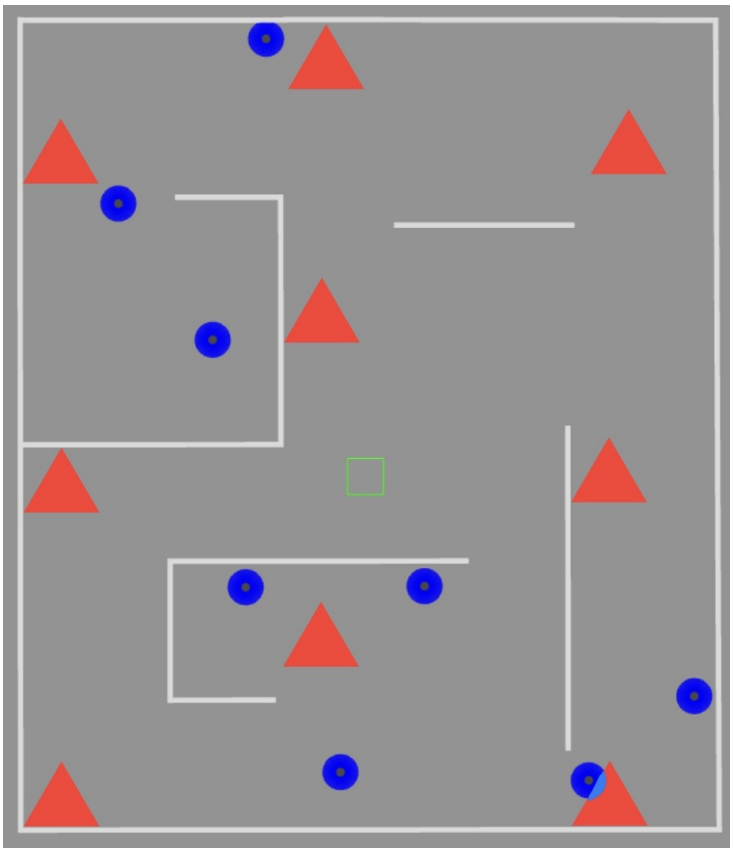

Figure 4.2: 8 TurtleBots3 model Waffle [17] from a top-down view. They are the black dots at the centre of the blue circles, while the blue circles indicate the radar sensors. Robots are in a random starting position inside the Arena. The green square indicates the origin of the ground ($x = 0, y = 0$). The red triangles are 9 scenes and the white lines are the walls.

4.1. Performance

To evaluate the performance of Swarm-SLAM, I simulate eight noisy robot's trajectories and I compare the solutions without (in Figure 4.3) and with (in Figure 4.4) inter-robot loop closure correction.

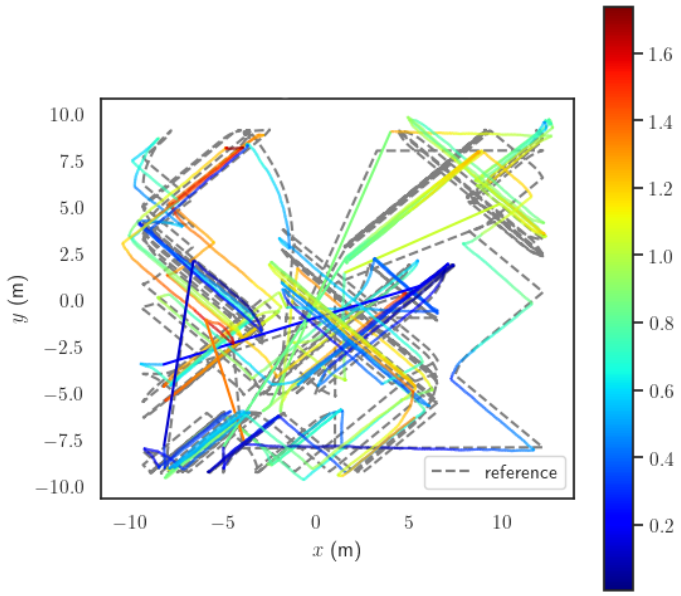

Figure 4.3: The trajectories of 8 reliable robots (coloured line) are unified in a pose graph and compared with the ground truth (dashed line), when noise is present in the odometry. The values of the parameters of the noisy odometry model described in Section 3.3 are: $\alpha_1 = 150.0$, $\alpha_2 = 160.0$, $\alpha_3 = 1.0$, and $\alpha_4 = 0.001$. The colour bar on the right indicates the amount of error at each point of the pose graph. The overall pose graph is consistently different from the ground truth and the portion of the graph in red corresponds to considerably high absolute positional error. The trajectories of the pose graph are aligned with the ground truth using Umeyama algorithm [19] as described in Section 4.2.

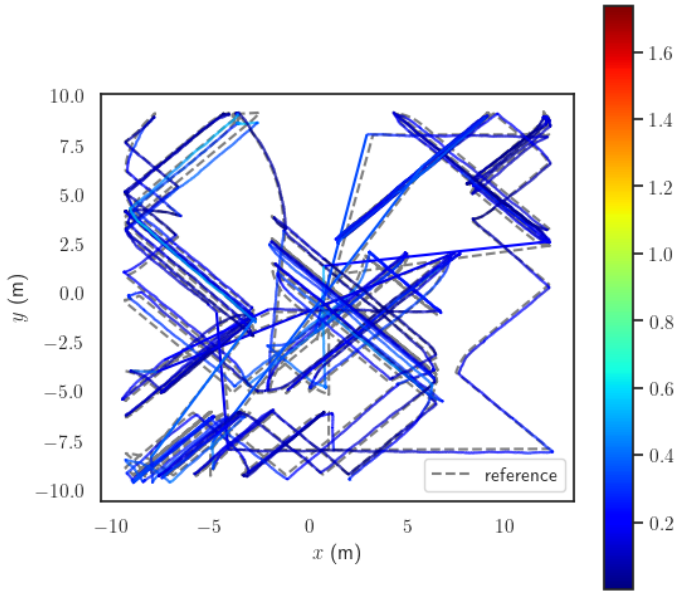

Figure 4.4: Evaluation of the Swarm-SLAM solution of eight noisy trajectories adjusted by the loop closures, represented in a pose graph (the coloured lines). The pose graph is compared with the ground truth (dashed line), as it is done in Figure 4.3. When Swarm-SLAM corrects the odometry the absolute positional error is much lower than in the case without Swarm-SLAM correction. The trajectories of the pose graph are aligned with the ground truth using Umeyama algorithm [19] as described in Section 4.2.

The pose graphs reported here are the final maps produced by Swarm-SLAM at the end of the simulation. However, the map's construction is an online process and maps are updated and improved in real-time throughout the experiment. In my thesis, I compare maps at the end of the experiments, i.e., after a pre-defined amount of time of robots' exploration and mapping. In this first example, it is clear how the algorithm is very effective in adjusting the trajectories that would otherwise be deformed by odometry errors. Indeed RMSE (see Section 4.2 for a detailed description of the metrics used), that is the quantity that best quantifies the total amount of error in the map with respect to the reference, is strongly reduced (see Figure 4.6). Figure 4.5 shows a comparison of the absolute positional error distributions in the case in which Swarm-SLAM for odometry correction is used and in the case in which it is not used.

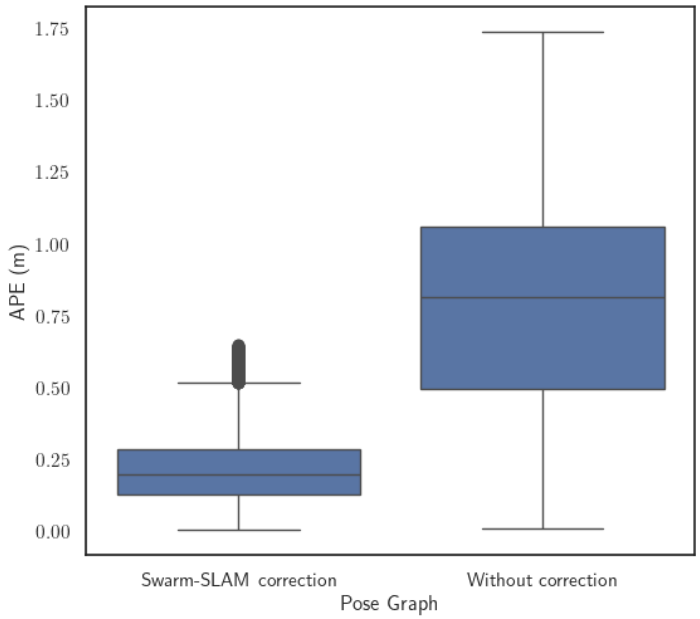

Figure 4.5: Representation of the comparison between two box plots of the Absolute Positional Error (APE) distribution. On the left, is depicted the APE distribution of a pose graph on which Swarm-SLAM correction through inter-robot loop closures is performed. While, on the right is represented the APE distribution for the pose graph of eight trajectories without Swarm-SLAM correction. The box plot displays the distribution and summary statistics of the data. In particular, the length of the blue box illustrates the spread of the middle 50% of the data, the lines that extend from the boxes indicates the variability of the data outside the upper and lower quartiles, the horizontal line inside the box represents the median of the data set. The outliers are represented as black dots. In my results there is a high number of statistical outliers due to points in which the APE is particularly high. A big concentration of these outliers takes the form of a vertical black strip in the plot. In numerical values, the RMSE with correction is 0.2340 (left), while without correction it is 0.8674 (right).

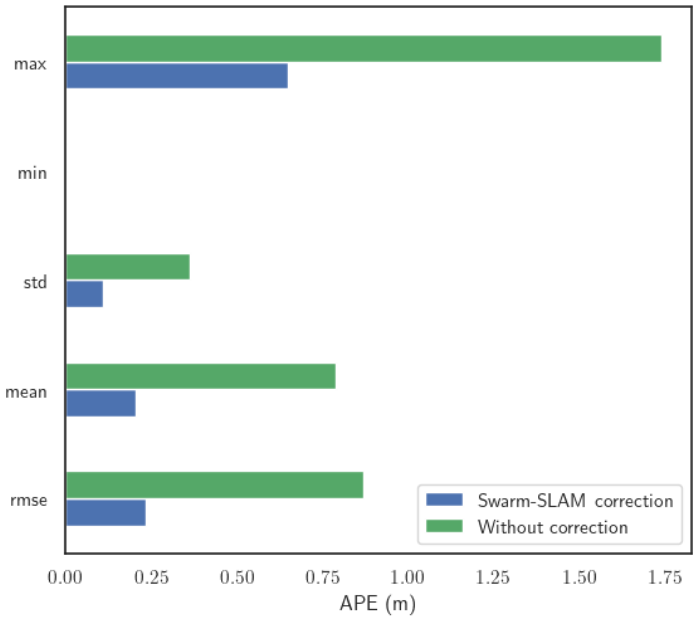

Figure 4.6: Representation of the statistics from the comparison between Swarm-SLAM correction and no correction. **max** indicates the maximum APE, **min** indicates the minimum APE, **std** is the standard error, **mean** is the mean over the APE distribution, **RMSE** is the Root Mean Square Error. It can be observed that all the metric errors have lower values in the solution of Swarm-SLAM. In particular, the RMSE, that is a good indicator of the overall pose graph quality, is much lower.

4.2. Robustness analysis

In this section, I present the results of my investigation into the effectiveness of the Byzantine fault-tolerant smart contract executed on the Toy-Chain, that I presented in Section 3.3. The study aims to determine whether the new method leads to a significant improvement in the rejection of Byzantine robots' input, compared to the original non-secured Swarm-SLAM. I refer to "secured Swarm-SLAM" when the Toy-Chain is active.

A robustness analysis is described in different scenarios, e.g., situations with different proportions of Byzantines over the total number of robots, or different Byzantine behaviours. Additionally, a technique to identify Byzantine robots in the swarm and to limit their "economic" power is discussed.

4.2.1. Metric errors

The measures that I use to quantify the map accuracy are presented in this paragraph. For the statistical analysis of the results I refer to four error indicators typically used in the SLAM literature: absolute positional error, root mean square error, standard error and mean error. All the metrics, maps and results reported in this thesis are evaluated using the Python package "evo", which is widely used for the evaluation of odometry and SLAM [19].

More precisely, when a pose graph is created, all the trajectories are merged into a global map composed of them. Inter-robot loop closures are the constraints that define the relative position between key positions of trajectories. A peculiar characteristic of Swarm-SLAM is that new local maps can be added in real-time to the swarm's map, once a link with a new participant is created, merging two subgraphs. Hence, if no inter-robot loop closures are created, no connection happens between individual maps and no optimisation takes place.

In Swarm-SLAM, the objective is to obtain nice global estimates of a 3D map; in my experiments, the goal is to obtain good trajectories in a global pose graph. Hence, in both cases, the most direct measure of accuracy is the misalignment between the 2D map composed of trajectories and the ground truth. Umeyama alignment [19] and correct scaling are performed as post-processing operations between the two 2D maps before the APE calculation. My results are independent of these operations, but this post-processing is important in order to maximally reduce other sources of error when calculating the metrics, isolating at best the topological difference between the two maps composed of trajectories.

- **Absolute Positional Error (APE):** it is a popular metric used to quantify the accuracy of location estimates in the map by measuring the difference between the reference positions (the ground truth poses) and the estimated ones (the maps poses). It is calculated for every pose in the map. It provides a scalar value that represents the magnitude of the error in metres.

  $APE = \sqrt{(x_{true} - x_{estimated})^2 + (y_{true} - y_{estimated})^2}$

- **Root Mean Square Error (RMSE):** It provides a good indication of the overall map accuracy.

  Let $\hat{X}_t$ be the estimated pose at time t, and X_t be the corresponding true pose.

  $RMSE = \sqrt{\frac{1}{N} \sum_{t=1}^{N} \|\hat{X}_t - X_t\|_2^2}$

  Here, N is the total number of poses in the map, $\|\cdot\|_2$ denotes the Euclidean norm.

When I compare results between two pose graphs obtained with different techniques, I also usually refer to the minimum (min) and the maximum (max) errors. They indicate the smallest and the biggest difference of the APE distribution over all the map, respectively. The mean error and the standard deviation are also calculated over the sample distribution.

Note 1: Due to the nature of my simulations, where inter-robot loop closures creation is based on robot poses acquired from Gazebo, small and uncontrollable differences in timestamps can lead to slightly different amount of loop closures generated for the same run. General results and conclusions remain valid in every simulation, but the same run is not always perfectly reproducible. This introduces a small variance in the number of inter-robot loop closures obtained, and it could be a problem if someone wants to obtain identical numerical results running multiple times the same simulation.

Note 2: The communication over the swarm participants is local in both the simulation level, and the blockchain security level. Hence, local knowledge is considered. The range of communication between robots is set to 4 metres.

Note 3: Each robot has the capability to perceive a scene when within a range of 4 metres.

4.2.2. Byzantine robots - Constant noise

The most trivial Byzantine behaviour is when a malicious or faulty robot add, intentionally or erroneously, a constant error in the loop closure calculation.
Here, simulation results for one run and different numbers of Byzantine agents are shown. The malicious or faulty behaviour is the addition of a constant 9.0 metres error on translations, at every loop closure calculation.

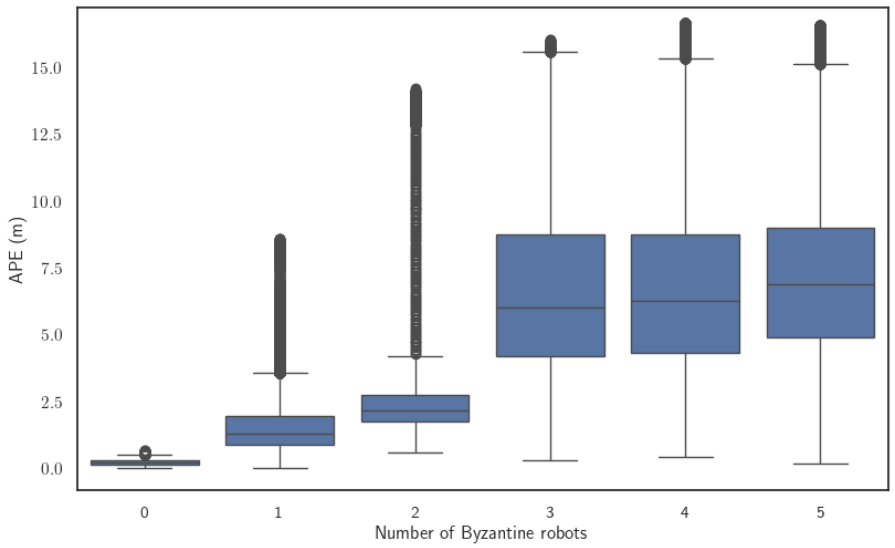

Figure 4.7: Performance in terms of APE of a non-secured Swarm-SLAM system in case Byzantine robots injecting constant noise are present. The overall APE is significantly bad. Moreover, the APE values increase noticeably with increasing number of Byzantine robots, which reaches values larger than 8 metres when just one Byzantine robot is present. Further details about this type of plot are discussed in the caption of Figure 4.5.

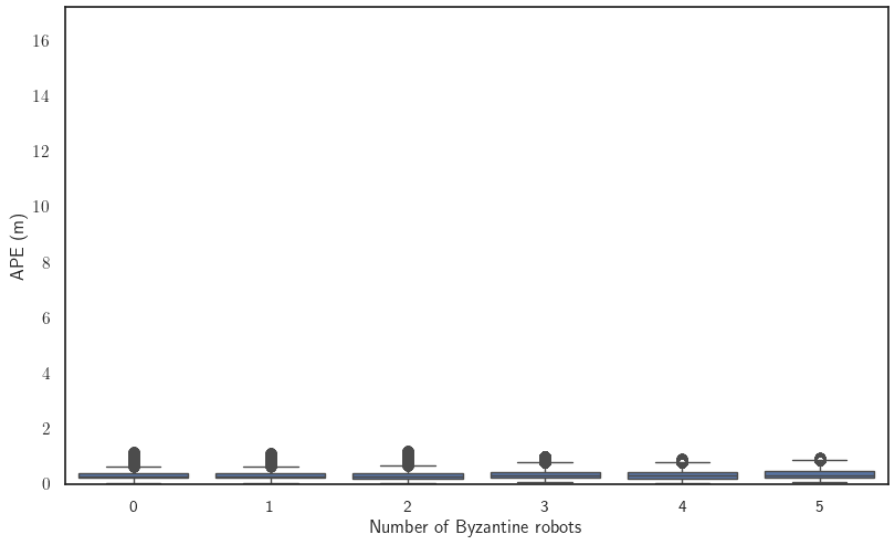

Figure 4.8: Secured Swarm-SLAM in case of constant noise perturbation. Notice how the APE is always very low. Moreover, the APE values never increase and the Byzantine behaviour is very well controlled.
Additionally, in general the overall error decreases when Byzantine robots are more. This corresponds to the fact that the final map is composed of fewer trajectories, so, the odometry error is controlled better and pose graph optimisation is accomplished more easily. An almost constant noise is ever present in the map because of my simulations include noise in the odometry. Further details about this type of plot are discussed in the caption of Figure 4.5.

As shown in Figure 4.7, in case of a non-secured systems, the error in the final 2D map tends to increase with the number of Byzantine robots, when the perturbation consists in constant noise injection. Clearly, this corresponds with the fact that the more Byzantine robots there are, the more wrong loop closures are used. Contrarily, when Swarm-SLAM is secured by the Toy-Chain's smart contract the final pose graph fits well the ground truth and the negative action of Byzantine robots is stopped. Figure 4.8 shows that the absolute positional error is always very low in secured Swarm-SLAM, compared to the non-secured Swarm-SLAM case. Figure 4.9 and Figure 4.10 report the statistics of the results for the metrics described in Section 4.2.1.

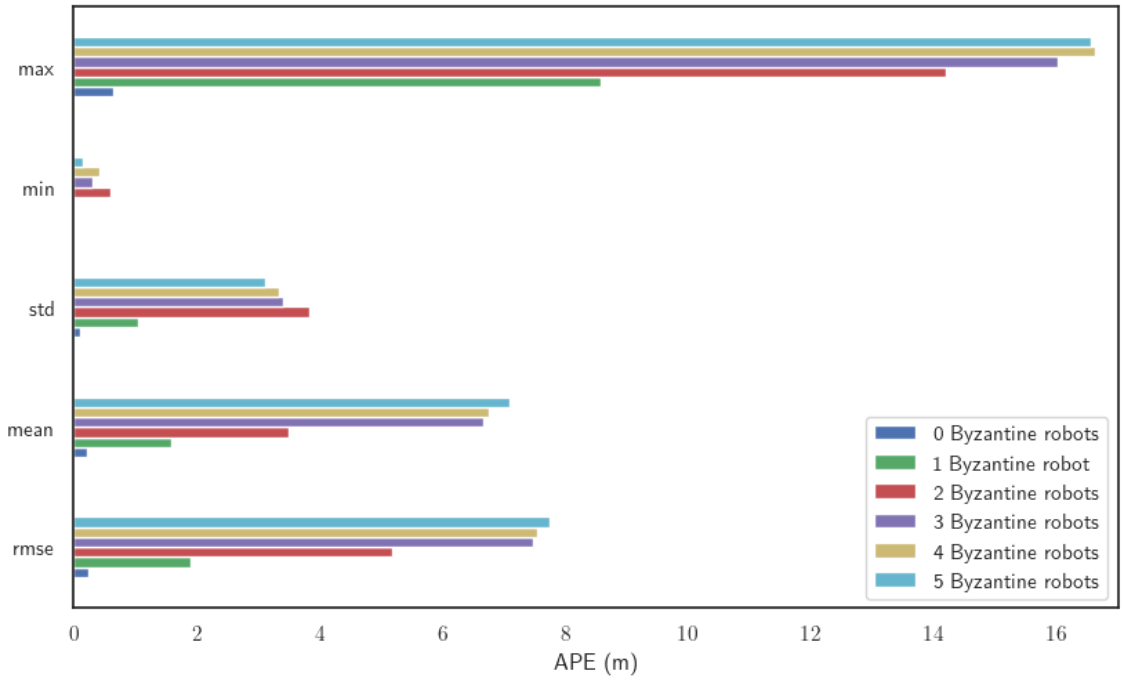

Figure 4.9: Non secured Swarm-SLAM. The figure depicts statistical results in case of constant error as Byzantine behaviour for different numbers of Byzantine robots in the system. The legend represents the number of Byzantine robots that are present in the system and associates to every case a specific colour in the bar plot. For example, colour blue means that the number of Byzantine robots is zero, while eight reliable robots are performing C-SLAM. On the x-axis is reported the Absolute Positional Error (APE) in metres for every case (0 , 1, 2, 3, 4, 5 Byzantine robots). On the y-axis are grouped together the results from different metric errors, computed measuring the APE of the pose graph with respect to the ground truth. **max** indicates the maximum APE, **min** indicates the minimum APE, **std** is the standard error, **mean** is the mean over the APE distribution, **RMSE** is the Root Mean Square Error.
It can be observed that the RMSE, that is a good indicator of the overall pose graph quality, monotonically increases with the number of Byzantine robots. Moreover, the maximum APE corresponds to several metres every time a Byzantine robot is participating in the SLAM procedure, leading to very high inaccuracy of the pose graph, and hence of the map. When there are no Byzantine robots the APE is low as expected, it is not zero because of the noise in the odometry.

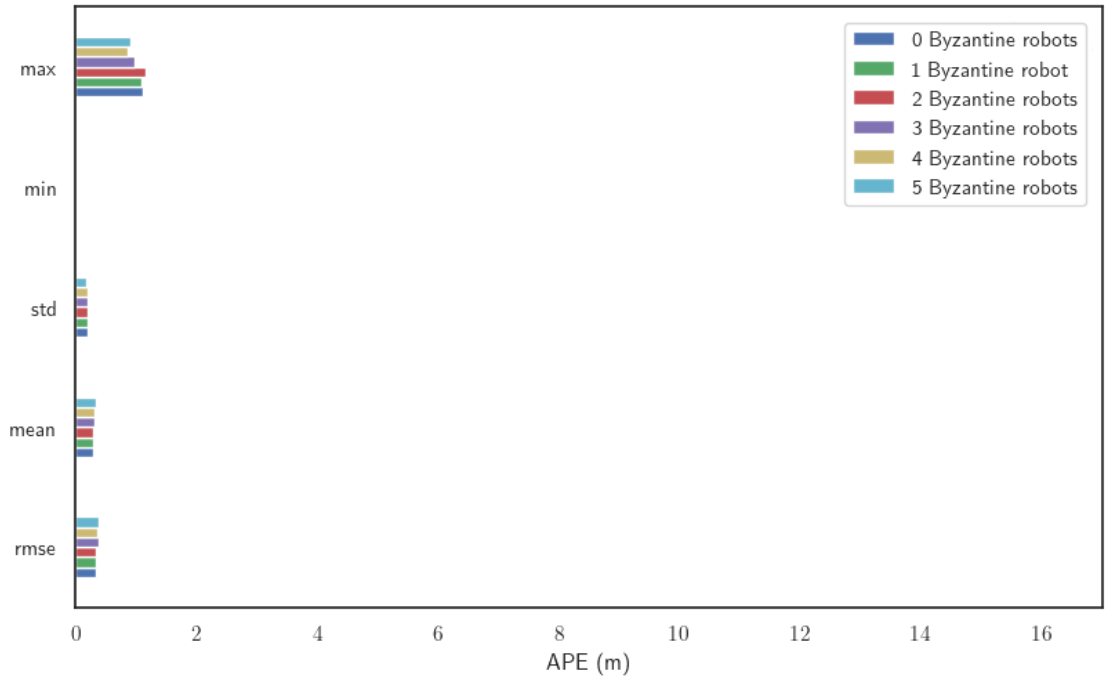

Figure 4.10: Secured Swarm-SLAM. The figure depicts statistical results in case of constant error as Byzantine behaviour for different numbers of Byzantine robots in the system. It can be observed that the APE is always very low. In particular the RMSE remains almost constant and around zero, independently on how many Byzantine robots are present. Further details about this type of plot are discussed in the caption of Figure 4.9.

To practically demonstrate the power of the blockchain based security protocol here discussed, a visual comparison between two Swarm-SLAM maps is reported in Figure 4.11 and Figure 4.12. A total of 5 Byzantines over 8 robots in the swarm is employed. It is clear how 5 Byzantine agents that send wrong inter-robot loop closures with a constant value of noise, completely destroy the map and lead to very poor map accuracy. Moreover, localisation is not possible at all and single honest individuals are negatively influenced by the collaboration. However, with the utilisation of the proposed smart contract, the swarm is able to overcome these limitations, even when malicious robots correspond to the majority (assuming no collusion between Byzantine robots).

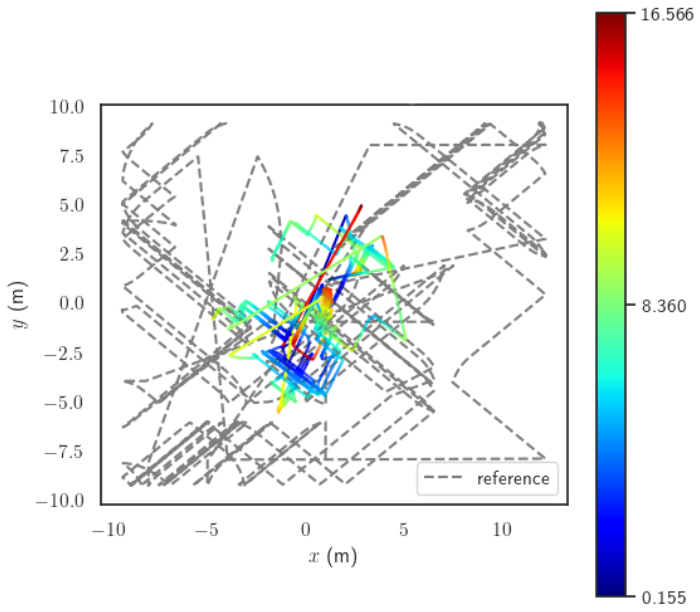

Figure 4.11: The pose graph composed by the trajectories of eight robots (coloured line) is compared with the ground truth (dashed line). The colour bar on the right indicates the amount of error at each point of the pose graph. Five Byzantine robots damage the non-secured Swarm-SLAM localisation, leading to peaks of more than 16 metres error in a map of 20×22 square metres. The trajectories of the pose graph are aligned with the ground truth using Umeyama algorithm [19] as described in Section 4.2.

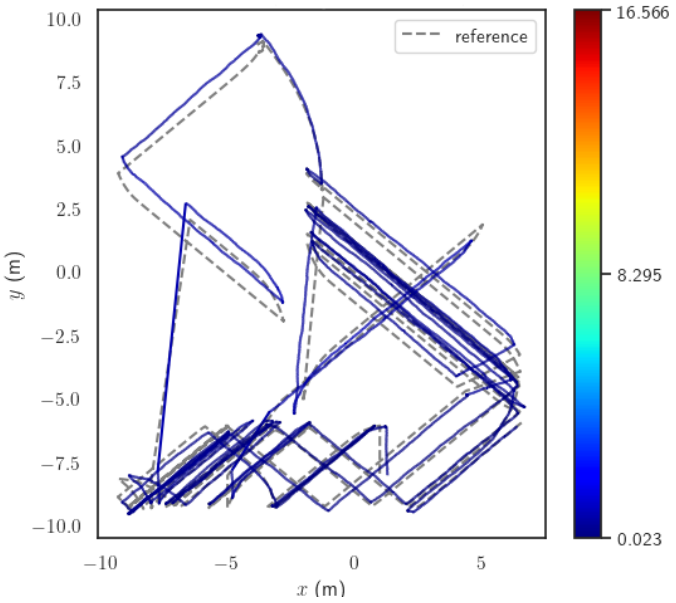

Figure 4.12: The pose graph composed by the trajectories of eight robots (coloured line) is compared with the ground truth (dashed line). The colour bar on the right indicates the amount of error at each point of the pose graph. Five Byzantine robots are present, but the secured Swarm-SLAM protocol is able to limit Byzantine behaviours, achieving very small accuracy errors. The trajectories of the pose graph are aligned with the ground truth using Umeyama algorithm [19] as described in Section 4.2.

It can be observed that the secured Swarm-SLAM framework generates smaller pose graphs compared to the complete ground truth of all the eight robots, in terms of number of trajectories, so in terms of pose graph lenght and number of keyframes—simply by looking at the difference between the ground truth of the non-secured Swarm-SLAM in Figure 4.11 and the secured Swarm-SLAM map of Figure 4.12. This is easily explained by examining the characteristics of Swarm-SLAM. Specifically, trajectories from Byzantine robots are not considered, thanks to the filtering action of the smart contract. This is possible because triangles cannot be validated when loop closures by Byzantines are part of them. Consequently, Byzantines' trajectories are not integrated into the global map. This provides two advantages: first, it prevents the system from incorrect constraints injection, and, second, it avoids the consideration of potentially unreliable trajectories. The latter aspect is crucial because, in the original non-secured Swarm-SLAM, any misbehaving robot can always merge its local map with the shared one, leading to uncontrolled results.

4.2.3. Byzantine robots - Random noise

Byzantine random noise injection can be more difficult to detect in collective systems with respect to the constant case, due to its stochasticity. Here, the Byzantine behaviour is the random perturbation of every erroneous loop closure proposal, again in their translational component. In particular, the random perturbation corresponds to the addition of a random error on translations drawn from a uniform distribution $\mathcal{U}[-9.0, 9.0]$ in meters.

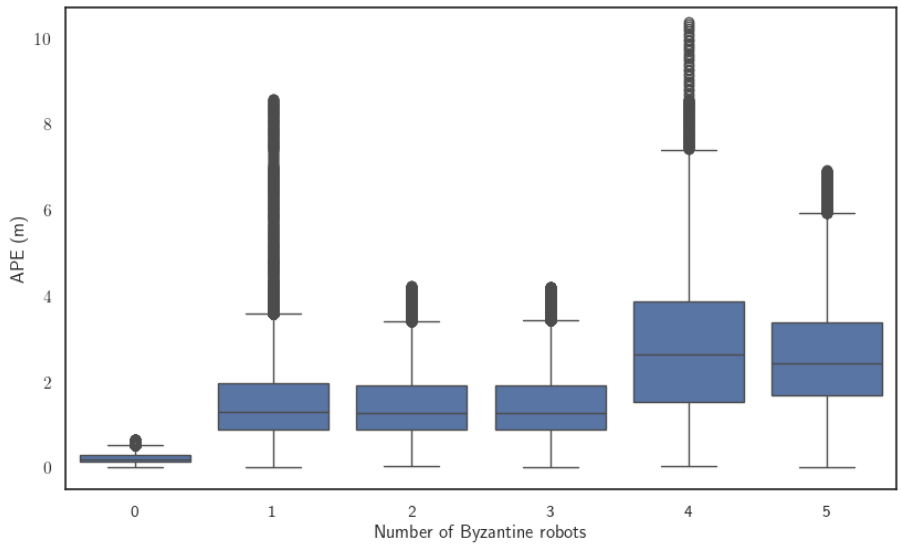

Figure 4.13: Non secured Swarm-SLAM in case of random noise perturbation. The APE values in case of one Byzantine are already very big with respect to the APE values when all robots are reliable. In particular, it is important to highlight the outliers in the APE distribution. Very big peaks of APE mean that somewhere in the map there is an error of several metres in the localisation. Further details about this type of plot are discussed in the caption of Figure 4.5.

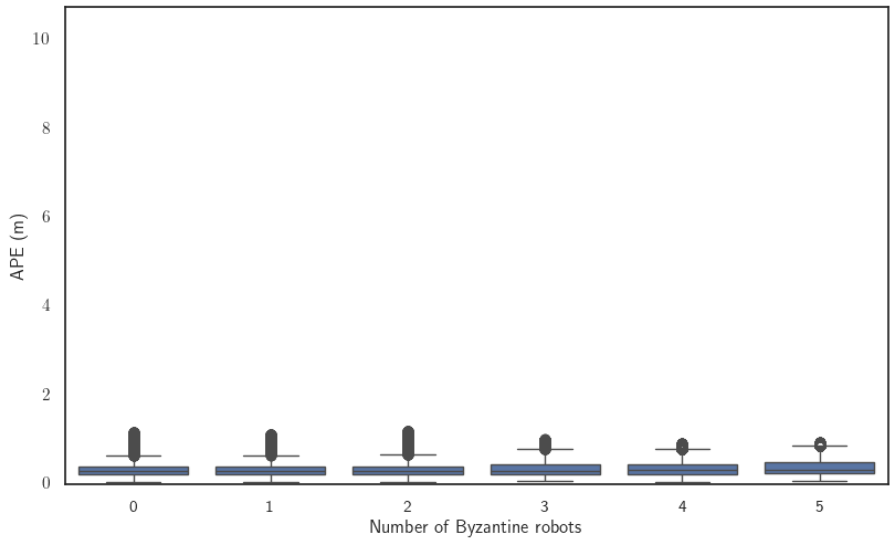

Figure 4.14: Secured Swarm-SLAM in case of random noise perturbation. Notice how the APE is always very low, moreover peaks of error are never present and Byzantine behaviour is very well controlled. Same further conclusion of the constant error case can be derived. Further details about this type of plot are discussed in the caption of Figure 4.5.

When no security checks are accomplished, the APE and RMSE errors are not monotonically increasing with the number of Byzantine robots, as shown in Figure 4.13. This is different with respect to the constant error case. The explanation is related to the experimental setup. When simulations with different numbers of unreliable robots are carried out, the added random error changes even for the same translations in different runs because of the different bunch of wrong loop closures. Sometimes, it leads to lower cumulative error even in the presence of higher numbers of erroneous loop closures. However, as for the constant case, Figure 4.14 shows that the absolute positional error is always very low in the secured Swarm-SLAM, compared to the non-secured Swarm-SLAM case. Figure 4.15 and Figure 4.16 report the statistics of the results for the metrics described in Section 4.2.1.

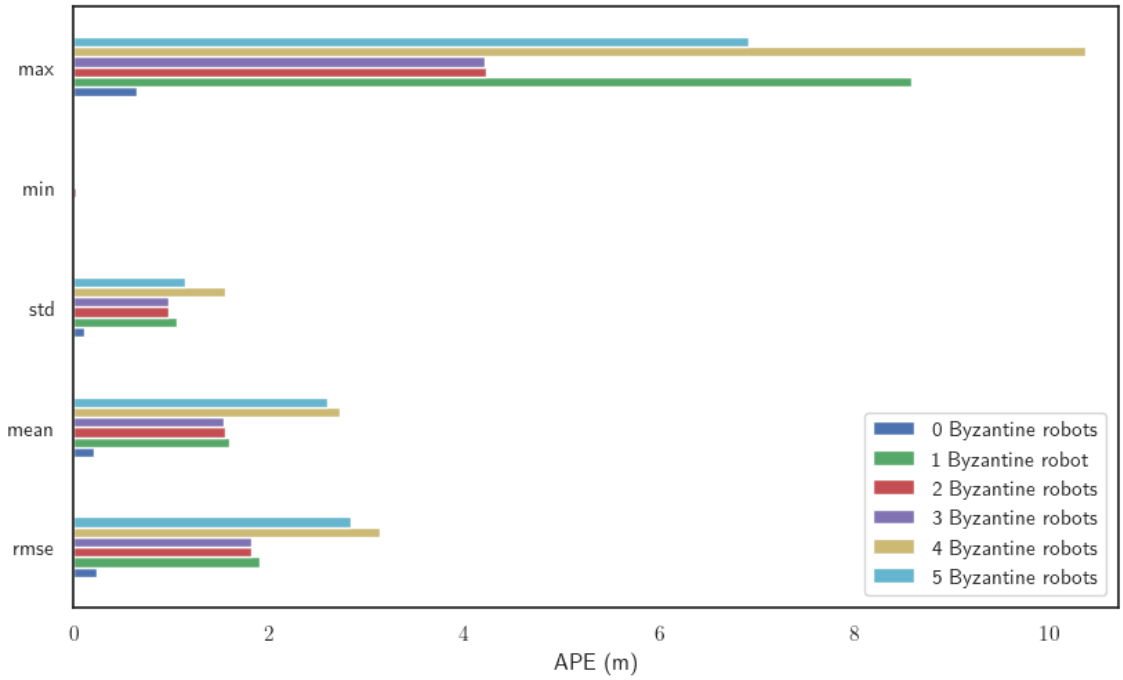

Figure 4.15: Non secured Swarm-SLAM. The figure depicts statistical results in case of random error as Byzantine behaviour for different numbers of Byzantine robots in the system. It can be observed that the APE is high every time a Byzantine robot is participating in the SLAM procedure. When there are no Byzantine robots the APE is low as expected, it is not zero because of the noise in the odometry. Further details about this type of plot are discussed in the caption of Figure 4.9.

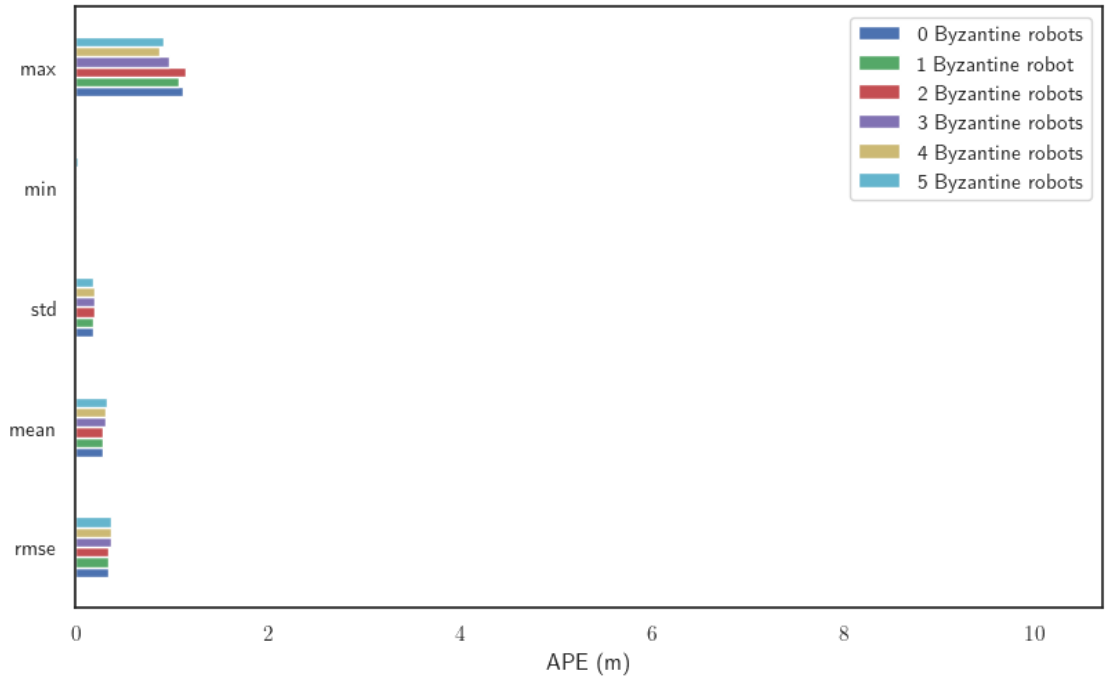

Figure 4.16: Secured Swarm-SLAM. The figure depicts statistical results in case of random error as Byzantine behaviour for different numbers of Byzantine robots in the system. It can be observed that the APE is always very low. In particular the RMSE remains almost constant and around zero, independently on how many Byzantine robots are present. Further details about this type of plot are discussed in the caption of Figure 4.9.

4.2.4. Statistical analysis

In this paragraph, a comprehensive statistical analysis is carried forward. In particular, five simulations for every case (no Byzantines, 1, 2, and 3 Byzantines) were recorded and their results aggregated. Here, the encoded negative behaviour is the random noise perturbation.
The analysis of test scores, the APE distribution, reveals a significant difference between the two frameworks. Swarm-SLAM equipped with the blockchain-based security protocol outperforms the standard Swarm-SLAM when more than zero Byzantines propose loop closures.

Note: A single artificial loop closure between honest robots and the Robot 0 (that is the robot in charge to perform the optimisation) is instantiated at the beginning of every simulation, when statistical analysis is studied. It allows to have a very brief moment of global communication between all the robots, allowing non-Byzantine participants to validate their first loop closures in order to obtain a global map from the beginning. Otherwise, if no links are generated between trajectories during the runs, pose graphs of different robots are not connected and the global map is divided into two submaps, leading to inconsistent results when comparing RMSE. This procedure ensures complete pose graphs even when connection between robots is not always guaranteed in short simulations.

In Figure 4.17, the RMSE when the smart contract is used (blue boxes) remains low despite the presence several Byzantine robots (in a swarm of eight robots in total). Instead, when security is not guaranteed (orange boxes) the error increases significantly and rapidly with the introduction of Byzantine robots. The APE error is never zero due to the presence of noise in the odometry that cannot be perfectly corrected. There is a difference between the two boxes in the case of all reliable robots because, even in this scenario, the two protocols work differently in the loop closure generation. In particular, in the non-secured system, the sender of the inter-robot loop closure is elected randomly at every meeting. In the secured Swarm-SLAM case, the smart contract forces the couples to elect the sender in a way that maximises the number of completed triangles. Remember that a triangle, to be considered so, must have all the arrows oriented in a cyclic way (either clockwise or counterclockwise).

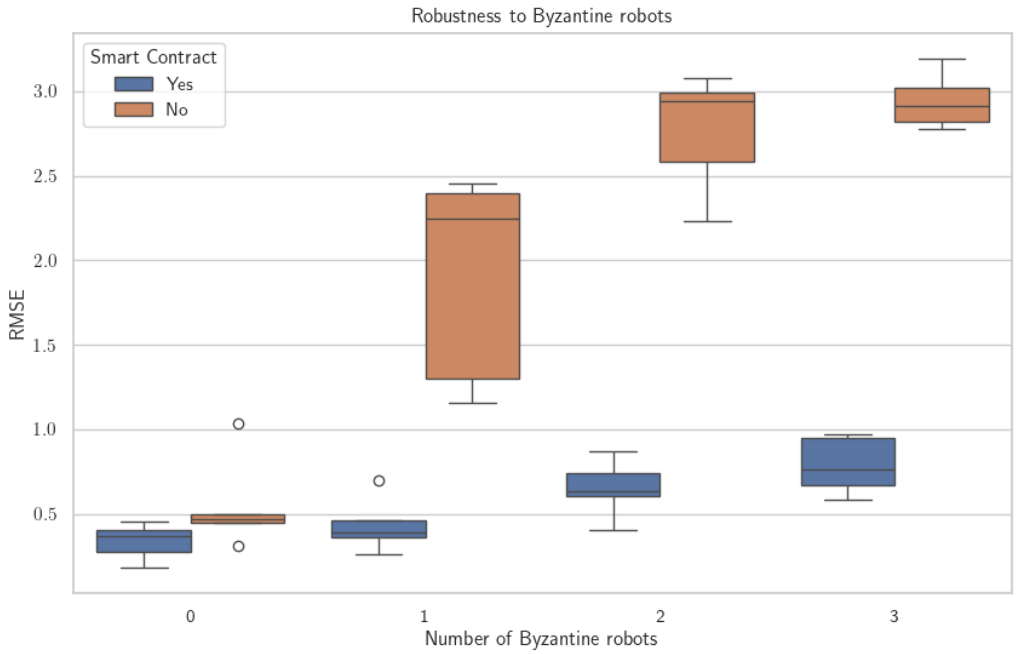

Figure 4.17: RMSE evaluation varying the number of robots. It is interesting to notice that, also in the case where the smart contract protects the system from wrong information injection, the RMSE increases with the number of Byzantine robots (blue boxes). This is directly related to the fact that the more Byzantine agents are present, the fewer inter-robot loop closures are accepted. In turn, a smaller number of loop closures leads to a tiny degradation of performance. Further details about this type of plot are discussed in the caption of Figure 4.5.

Efficiency refers to the ability of Swarm-SLAM to optimise at best the pose graph in a specific time frame, to accomplish real-time performance requirements. Online map generation is important in SLAM systems and the trade-off between efficiency and security have to be considered.

In a standard Swarm-SLAM application, all the available information are immediately used in the optimisation, but I demonstrated that this can lead to disrupting consequences in localisation and mapping. The security protocol, that in this work is introduced, significantly improves security and leads to the creation of good maps even in case of Byzantine robots. In order to accomplish that, inter-robot loop closures enter in a pending state until their validation, and only a part of them is published and used in a precise time frame. In Figure 4.18, the fact that with more malicious or faulty robots less loop closures are available to optimise the map during the simulations is shown.

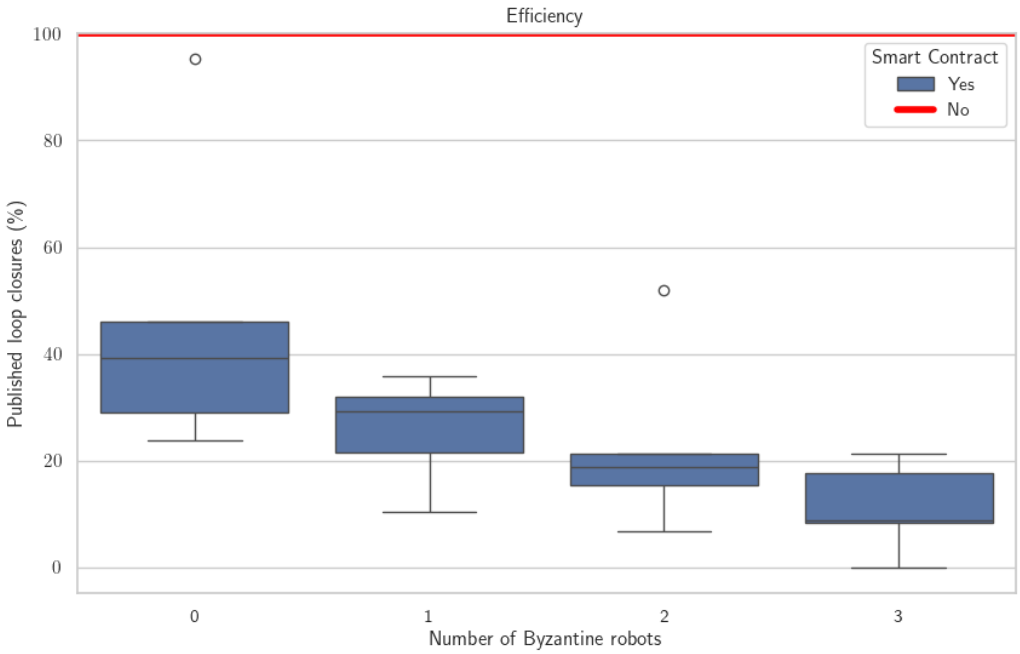

Figure 4.18: Percentage of approved inter-robot loop closures.

Reputation measures the contribution of the robots in achieving the swarm's goal through a scalar value. The more inter-robot loop closures whose Robot i is the sender are approved, the higher reputation Robot i has. Reputation rewards are also determined by the number of triangles in which a loop closure is involved, higher social confirmation leads to a bigger reward. This leads to the conclusion that the robots with reputation equal to zero are most probably Byzantines, since they always have zero reputation as in Figure 4.19. In this study I considered only robots with static behaviours. If the Byzantine behaviour is considered a dynamic property, high reputation cannot be directly associated with a honest robot behaviour, but it still indicate the positive contribution of a robot in achieving the swarm's goal. Robots that achieved a non-null reputation could be Byzantine robots if their behaviour changed over time from honest to malicious and they were rewarded for correct loop closures creation. Regardless of the specific type of adversarial attack and the reputation of the robots at any given moment, one consistent aspect remains: incorrect inter-robot loop closures are consistently rejected.

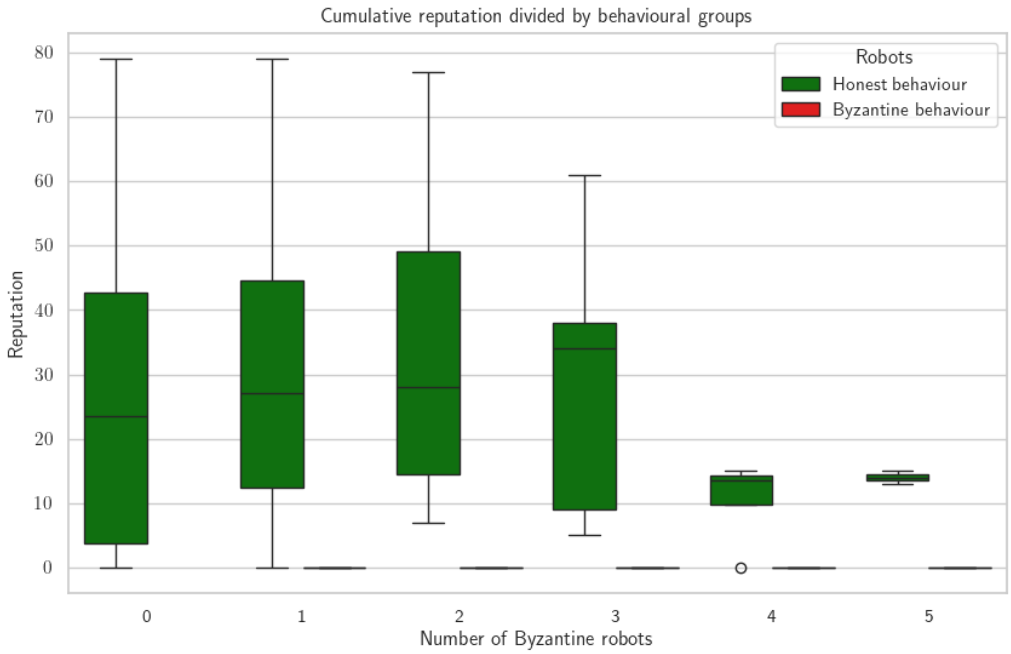

Figure 4.19: Robots are categorised into two groups: those exhibiting good behaviour (green) and those with Byzantine behaviour (red). The goal here is to demonstrate that the overall reputation of the Byzantine robots consistently remains zero. Hence, the sum of the reputation values of all the robots belonging to the same category is calculated. Then, for every scenario (0 Byz., 1 Byz., 2 Byz., and 3 Byz.), the two cumulative values are compared on the x-axis. As expected, the total reputation of all the good robots decreases as there are more Byzantines in the swarm. It is noteworthy that the colour red never appears inside the plot because the boxes associated with the Byzantine robots, positioned to the right of every green box, consistently display zero values.

4.3. Security and efficiency trade-off

A higher security level corresponds to fewer loop closures approved in the same time period, resulting in worse performance. This is the trade-off that must be evaluated when applying the security protocol developed during this work.

In general, Byzantine fault-tolerant Swarm-SLAM is thought to be scalable and used in highly populated environments, where redundancy is ever present and local connections are frequent. These are key characteristics of swarm robotics, and they represent the assumption on which I built this framework.

Based on these considerations, I expect that in long-term swarm robotics operations the

number of triangles that are validated is very high. However, this could also introduce the possibility of more complex malicious behaviours in the Byzantine robots. In particular, coalitions between malicious robots.
In this context, the security level parameter introduced in Section 3.2.2 can be useful to tune the security to the needed level and the scenario characteristics. Based on the system parameters, the security level could easily reach level 10 or 20 for the most confirmed inter-robot loop closures.

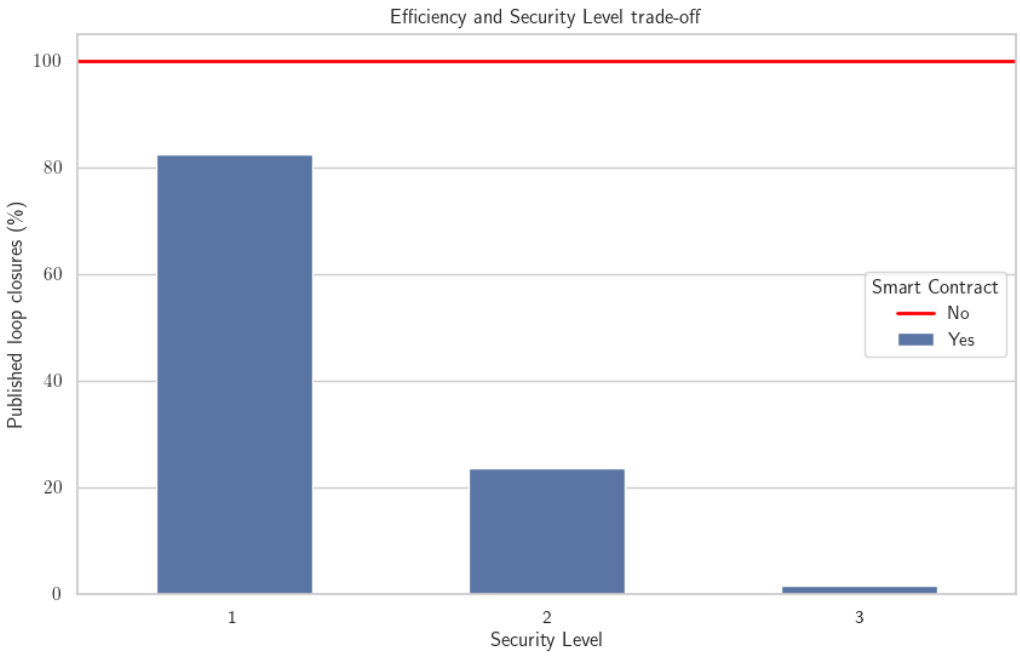

Figure 4.20: In this figure, the y-axis represents the percentage of inter-robot loop closures, while the x-axis corresponds to the reported security levels. The data are evaluated after a 45-minute-long run. In this experiment, local communication is considered within a range of 5 metres. It is important to notice that the simulation was stopped after 40 minutes of simulated time, that is when the first set of loop closures reached level 3, with longer runs higher levels of security would certainly be reached.

Figure 4.20 reports a clear example of how the number of inter-robot loop closures is strongly reduced, along with the threshold value for the loop closure security acceptance.

5 | Further considerations and conclusion

In this final chapter, the key findings of the research are synthesised to provide a comprehensive understanding of:

1. The robustness problem in C-SLAM systems when Byzantine robots participate to collective actions.
2. A new Byzantine fault-tolerant framework for Swarm-SLAM secured by a blockchain-based smart contract.

This section revisits the research questions and objectives, summarising the main contributions of the study.

5.1. Future work

Building on the current study, there are several directions for future research. In this thesis I worked under the assumption that the robot in charge to perform the pose graph optimisation was not Byzantine and I used a very simple custom blockchain, the Toy-Chain. These are two important constraints that still have to be solved in order to have fully autonomous and secure Swarm-SLAM systems. Discussing these two extensions of my work, I provide a roadmap for researchers interested in further exploring Swarm-SLAM Byzantine fault-tolerant architectures.

5.1.1. Secure optimisation management

In Swarm-SLAM, pose graph optimisation is a key capability of robots for solving the simultaneous localisation and mapping problem. It refers to the computational ability of a robot to solve a computationally hard operation, in order to obtain improved maps based on new constraints. It is resource intensive and it is usually solved by a single robot. Therefore, it remains a "central" role in a system that is thought to be decentralised.

Such an important issue should never be addressed as a single robot duty, especially in a swarm of robots. This leads to the creation of a single point of failure in the entire system, limiting the swarm capabilities. In particular, in Swarm-SLAM the robot with the lowest ID is the one in charge to carry on pose graph optimisation before sharing the result with the other swarm participants; if it were a Byzantine robot the entire SLAM operation could be compromised.

A promising new research direction, strictly related to my study, is to investigate a new "intelligent" pose graph optimisation management strategy. Most important is the introduction of redundancy in this fundamental procedure, allowing multiple robots to perform it, such that in case of Byzantine robots the information is not lost or altered. Then, I suggest to develop techniques to manage which robots are elected to tackle this, based on their reputation, their computational capabilities, and their communication possibilities at the current moment.

Here, is where blockchain comes into help. New consensus algorithms could be implemented through smart contracts in order to validate the most reliable pose graph, based on voting mechanisms. Reputation of agents would consequently be modified based on their performance.
The biggest challenge following this approach could be to find a compact way to encode pose graphs information into a blockchain transaction, and to identify a metric for comparing these abstract information.

5.1.2. Ethereum blockchain

Regarding the limitations of my blockchain implementation, I used the Toy-Chain, a simple highly customisable blockchain that allows to code smart contracts. It works following the same basic functionalities of real blockchains. However, blockchains like Ethereum are much more complex in their working principles, in particular from the point of view of cryptographic security protocols.
Concerning my purpose, cryptographically secure structures were not important, since experiments were mainly used as proof of concept. However, if a reader wants to use a blockchain secured Swarm-SLAM system in practical applications, a universally embraced blockchain platform should be used. Given the promising outcomes observed in the utilisation of Ethereum in swarm robotics [53], which have demonstrated the effective management of computational requirements by a swarm of robots, an interesting extension to this work could be the realisation of an Ethereum-based Swarm-SLAM framework.

5.2. Conclusion

The study revealed that the C-SLAM literature still lacks of frameworks and protocols to address security issues related to malicious or unreliable robots in the swarm, shedding light on different subproblems related to the inter-robot loop closures calculation.
In particular, a wrong loop closure directly affect the pose graph optimisation result, leading to potentially very incorrect maps. I first shown these dangerous consequences of a Byzantine attack. I then carried on a comprehensive theoretical analysis on the possible points of failure of a specific C-SLAM framework, Swarm-SLAM, and I proposed a practical and effective method to solve the principal challenges. Swarm-SLAM testing results aligned with the previous research on this very new field, emphasising the significance and the need of scalable and secure architecture with robustness properties.

This research contributes to a first attempt to address the presence of the Byzantine robots in C-SLAM systems. These contributions include the first integration of a custom blockchain, i.e. the Toy-Chain, into a Swarm-SLAM architecture. Moreover, the development of a blockchain-based smart contract that ensures reliable maps, maintaining good real-time performances is a ready-to-use framework for future studies. This study try to fill the gaps identified in the first works that paved the way for SLAM with robot swarms.

It is crucial to acknowledge the limitations of this study. My thesis work aimed to demonstrate that blockchain technology, which has been shown to be a successful instrument to limit Byzantine behaviours in swarm robotics [52] [53] [62] [54], is also an effective solution to improve security in Swarm-SLAM. Hence, I focused on the design and development of the Toy-Chain's smart contract algorithm and evaluated the results obtained from a very simple simulation setup. This work currently lacks extensive application of the Swarm-SLAM framework in real-world experiments, where the front end is not limited to simulation alone. These limitations present opportunities for future research to develop practical blockchain-based Byzantine fault-tolerant Swarm-SLAM protocols for real robots.

In conclusion, this thesis has advanced our understanding of security vulnerabilities in C-SLAM systems by analysing the state-of-the-art framework, and proposed an effective solution to the problem of the creation of inter-robot loop closures by Byzantine robots. The findings not only highlight some problems not addressed yet in C-SLAM, but also have practical implications for secured Swarm-SLAM implementations. As we move forward in SLAM systems development, it is essential to study secure and robust architectures to continue advancing the field and addressing the complex challenges posed by robot swarms subjected to Byzantine robots.

Bibliography

[1] J. Aulinas, Y. Petillot, J. Salvi, and X. Llado. The SLAM problem: a survey. *Frontiers in Artificial Intelligence and Applications*, 184:363–371, 2008. doi: 10.3233/978-1-58603-925-7-363.

[2] N. Ayache and O. Faugeras. Building, registrating and fusing noisy visual maps. *The International Journal of Robotics Research*, 7(6):45–65, 1988.

[3] G. Beni. The concept of cellular robotic system. *Proceedings IEEE International Symposium on Intelligent Control 1988*, pages 57–62, 1988.

[4] M. Bujanca, X. Shi, M. Spear, P. Zhao, B. Lennox, and M. Lujan. Robust SLAM Systems: Are We There Yet? *arXiv preprint*, 2021.

[5] Y. Chang, K. Ebadi, C. E. Denniston, M. F. Ginting, A. Rosinol, A. Reinke, M. Palieri, J. Shi, A. Chatterjee, B. Morrell, A. Agha-mohammadi, and L. Carlone. LAMP 2.0: A Robust Multi-Robot SLAM System for Operation in Challenging Large-Scale Underground Environments. *arXiv preprint*, 2022.

[6] 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):74, 2023. doi: 10.3390/machines11060653.

[7] T. Cieslewski, S. Choudhary, and D. Scaramuzza. Data-Efficient Decentralized Visual SLAM. *IEEE International Conference on Robotics and Automation (ICRA)*, page 8, 2018.

[8] 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.

[9] J. L. Crowley. World modeling and position estimation for a mobile robot using ultrasonic ranging. *Proceedings, 1989 International Conference on Robotics and Automation*, pages 674–680 vol.2, 1989.

[10] K. J. Doherty, D. M. Rosen, and J. J. Leonard. Spectral Measurement Sparsification for Pose-Graph SLAM. *arXiv preprint*, 2022.

[11] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006. doi: 10.1109/MCI.2006.329691.

[12] M. Dorigo, G. Theraulaz, and V. Trianni. Swarm Robotics: Past, Present, and Future [Point of View]. *Proceedings of the IEEE*, 109(7):1152–1165, 2021. doi: 10.1109/JPROC.2021.3072740.

[13] J. R. Douceur. The Sybil Attack. In *International Workshop on Peer-to-Peer Systems*, 2002.

[14] H. Duan, M. Huo, and Y. Fan. From animal collective behaviors to swarm robotic cooperation. *National Science Review*, 10(5):nwad040, 2023. doi: 10.1093/nsr/nwad040.

[15] 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.

[16] J. J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-Scale Direct Monocular SLAM. In *European Conference on Computer Vision*, 2014.

[17] O. S. R. Foundation. Turtlebot3. URL `https://www.turtlebot.com/turtlebot3/`.

[18] 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.

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

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

[21] B. Huang, J. Zhao, and J. Liu. A Survey of Simultaneous Localization and Mapping. *CoRR*, 1909, 2019.

[22] 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*, 7(2):1150–1157, 2022. doi: 10.1109/LRA.2021.3138156.

[23] E. Hunt and S. Hauert. A checklist for safe robot swarms. *Nature Machine Intelligence*, 2, 2020. doi: 10.1038/s42256-020-0213-2.

[24] IBM. What is blockchain technology?, 2023. URL `https://www.ibm.com/topics/blockchain`.

[25] M. Kegeleirs, D. Garzón Ramos, and M. Birattari. Random Walk Exploration for Swarm Mapping. In *Towards Autonomous Robotic Systems*, pages 211–222, Cham, 2019. Springer.

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

[27] 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)*, volume 3, pages 2149–2154 vol.3, 2004. doi: 10.1109/IROS.2004.1389727.

[28] P.-Y. Lajoie and G. Beltrame. Swarm-SLAM : Sparse Decentralized Collaborative Simultaneous Localization and Mapping Framework for Multi-Robot Systems. *arXiv preprint*, 2023.

[29] 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.

[30] 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.

[31] 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(1):971–1000, 2022. doi: 10.55417/fr.2022032.

[32] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Langanguages and Systems*, 4(3):382–401, 1982. doi: 10.1145/357172.357176.

[33] M. Lourakis and A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Transactions on Mathematical Software*, 36:1–30, 2009.

[34] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall. Robot Operat-

ing System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66): eabm6074, 2022. doi: 10.1126/scirobotics.abm6074.

[35] MathWorks. SLAM Processing Flow, 2023. URL `https://www.mathworks.com/discovery/slam.html`.

[36] MRPT. Probabilistic motion models, 2023. URL `https://docs.mrpt.org/reference/latest/tutorial-motion-models.html`.

[37] R. Mur-Artal, J. Montiel, and J. Tardos. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31:1147 – 1163, 10 2015. doi: 10.1109/TRO.2015.2463671.

[38] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. *Electronic document available at http://www.bitcoin.org*, pages 1–11, 2008.

[39] 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. IEEE, 2009.

[40] 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.

[41] 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 *LNCS*, pages 3–15, Cham, Switzerland, 2020. Springer. doi: 10.1007/978-3-030-60376-2_1.

[42] J. A. Placed, J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos. A Survey on Active Simultaneous Localization and Mapping: State of the Art and New Frontiers. 2023.

[43] A. Prorok, M. Malencia, L. Carlone, G. S. Sukhatme, B. M. Sadler, and V. Kumar. Beyond Robustness: A Taxonomy of Approaches towards Resilient Multi-Robot Systems. *arXiv preprint*, 2021.

[44] D. Rakita. Ethereum Virtual Machine (EVM), 2023. URL `https://ethereum.org/en/developers/docs/evm`.

[45] A. Reina. Robot teams stay safe with blockchains. *Nature Machine Intelligence*, 2: 240–241, 2020. doi: 10.1038/s42256-020-0178-1.

[46] A. Reina, T. Njougouo, E. Tuci, and T. Carletti. Studying speed-accuracy trade-offs

in best-of-n collective decision-making through heterogeneous mean-field modelling. *arXiv*, 2310.13694, 2023. doi: https://doi.org/10.48550/arXiv.2310.13694.

[47] A. Reina, R. Zakir, G. De Masi, and E. Ferrante. Cross-inhibition leads to group consensus despite the presence of strongly opinionated minorities and asocial behaviour. *Communications Physics*, 6:236, 2023. doi: 10.1038/s42005-023-01345-3.

[48] D. Rodriguez-Losada, F. Matia, and A. Jimenez. Local maps fusion for real time multirobot indoor simultaneous localization and mapping. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 2, pages 1308–1313. IEEE, 2004.

[49] S. Saeedi, M. Trentini, M. Seto, and H. Li. Multiple-robot simultaneous localization and mapping: A review. *Journal of Field Robotics*, 33(1):3–46, 2016.

[50] P. Schmuck, T. Ziegler, M. Karrer, J. Perraudin, and M. Chli. COVINS: Visual-Inertial SLAM for Centralized Collaboration. *arXiv preprint*, 2021.

[51] R. C. Smith and P. Cheeseman. On the Representation and Estimation of Spatial Uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1986. doi: 10.1177/027836498600500404.

[52] V. Strobel, E. Castelló Ferrer, and M. Dorigo. Managing Byzantine Robots via Blockchain Technology in a Swarm Robotics Collective Decision Making Scenario. In *Proceedings of 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.

[53] 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*, 7:54, 2020. doi: 10.3389/frobt.2020.00054.

[54] V. Strobel, A. Pacheco, and M. Dorigo. Robot swarms neutralize harmful Byzantine robots using a blockchain-based token economy. *Science Robotics*, 8(79):eabm4636, 2023. doi: 10.1126/scirobotics.abm4636.

[55] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[56] 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. *arXiv preprint*, 2021.

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

[58] G. Wood. Ethereum: A Secure Decentralized Generalised Transaction Ledger. *Ethereum Foundation*, pages 1–41, 2014.

[59] 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.

[60] R. Zakir, M. Dorigo, and A. Reina. Robot Swarms Break Decision Deadlocks in Collective Perception through Cross-inhibition. In *Swarm Intelligence (ANTS 2022)*, volume 13491 of *LNCS*, pages 209–221. Springer, Cham, 2022. doi: 10.1007/978-3-031-20176-9_17.

[61] Y. Zhang, Y. Wu, K. Tong, H. Chen, and Y. Yuan. Review of Visual Simultaneous Localization and Mapping Based on Deep Learning. *Remote Sensing*, 15(11), 2023. doi: 10.3390/rs15112740.

[62] 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. In *Proceedings of the 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2023)*, page in press. IEEE, 2023.

[63] Z. Zibin, X. Shaoan, D. Hongning, C. Xiangping, and W. Huaimin. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. *IEEE*, pages 1–9, 2017.

A | The Smart Contract

A snippet of Python code that constitutes the custom smart contract I developed is reported below.

All the material used to run the experiments is public available at my GitHub page:

- Smart Contract and ROS2 Toy-Chain integration: https://github.com/clmoro/toychain-ROS2
- Experimental setup: https://github.com/clmoro/swarm-SLAM-ROS2

Listing A.1: The Smart Contract

```
# Smart Contract function
def apply_validation(self, LC_Descriptor, LC_ID_R, LC_Odomx_R, LC_Odomy_R,
    LC_Keyframe_R, LC_ID_S, LC_Odomx_S, LC_Odomy_S, LC_Keyframe_S, LC_dx, LC_dy,
    LC_SCENE):

# Custom constants of the Smart Contract
bound = 0.0001
security_level = 1

new_triangle = True
new_triangle_i = True
new_triangle_j = True
new_triangle_k = True

# New Loop Closure registration
self.candidate_LC['LC_Descriptor'].append(LC_Descriptor)
self.candidate_LC['LC_ID_R'].append(LC_ID_R)
self.candidate_LC['LC_Keyframe_R'].append(LC_Keyframe_R)
self.candidate_LC['LC_ID_S'].append(LC_ID_S)
self.candidate_LC['LC_Keyframe_S'].append(LC_Keyframe_S)
self.candidate_LC['LC_dx'].append(LC_dx)
self.candidate_LC['LC_dy'].append(LC_dy)
self.candidate_LC['LC_SCENE'].append(LC_SCENE)
self.candidate_LC['LC_Security'].append(0)

# Triangles construction
for i in range(len(self.candidate_LC['LC_SCENE'])):
 for j in range(len(self.candidate_LC['LC_SCENE'])):
```

```
27        for k in range(len(self.candidate_LC['LC_SCENE'])):
28         # Check for the scene matching and No mutual edges between two participants (to
               avoid collusions)
29         if((self.candidate_LC['LC_SCENE'][i] == self.candidate_LC['LC_SCENE'][j] ==
               self.candidate_LC['LC_SCENE'][k] and i != j
30         and i != k and j!= k) and (self.candidate_LC['LC_ID_S'][i] !=
               self.candidate_LC['LC_ID_S'][j] !=
31         self.candidate_LC['LC_ID_S'][k]) and (self.candidate_LC['LC_ID_R'][i] !=
               self.candidate_LC['LC_ID_R'][j] !=
32         self.candidate_LC['LC_ID_R'][k])):
33          # Check a complete triangle has not already been approved
34          if(not(self.candidate_LC['LC_Descriptor'][i] in self.validated_LC['Descriptor']
                and self.candidate_LC['LC_Descriptor'][j] in self.validated_LC['Descriptor']
                and self.candidate_LC['LC_Descriptor'][k] in
                self.validated_LC['Descriptor'])):
35           # This check is for the Head-Tail correspondence in every possible vertice
                condition of the triangle
36           if(self.candidate_LC['LC_ID_S'][i] == self.candidate_LC['LC_ID_R'][j] and
                self.candidate_LC['LC_ID_S'][j] == self.candidate_LC['LC_ID_R'][k] and
                self.candidate_LC['LC_ID_S'][k] == self.candidate_LC['LC_ID_R'][i] and
                self.candidate_LC['LC_Keyframe_S'][i] ==
                self.candidate_LC['LC_Keyframe_R'][j] and
                self.candidate_LC['LC_Keyframe_S'][j] ==
                self.candidate_LC['LC_Keyframe_R'][k] and
                self.candidate_LC['LC_Keyframe_S'][k] ==
                self.candidate_LC['LC_Keyframe_R'][i]):
37            # This check is for every possible combination without taking into account the
                   direction of the trasformation. However, it will result True iff the
                   arrows have coherent circular direction
38            if((abs(self.candidate_LC['LC_dx'][i] + self.candidate_LC['LC_dx'][j] +
                   self.candidate_LC['LC_dx'][k]) < bound) and
                   (abs(self.candidate_LC['LC_dy'][i] + self.candidate_LC['LC_dy'][j] +
                   self.candidate_LC['LC_dy'][k]) < bound)):
39             # Possibly increase the security parameter of the approved LCs
40             for z in range(len(self.triangles)):
41              if(self.candidate_LC['LC_ID_S'][i] in self.triangles[z] and
                    self.candidate_LC['LC_ID_S'][j] in self.triangles[z]
42              and self.candidate_LC['LC_ID_S'][k] in self.triangles[z]):
43               if(self.candidate_LC['LC_Descriptor'][i] in self.triangles[z]):
44                new_triangle_i = False
45               if(self.candidate_LC['LC_Descriptor'][j] in self.triangles[z]):
46                new_triangle_j = False
47               if(self.candidate_LC['LC_Descriptor'][k] in self.triangles[z]):
48                new_triangle_k = False
49               if(self.candidate_LC['LC_Descriptor'][i] in self.triangles[z] and
                    self.candidate_LC['LC_Descriptor'][j] in self.triangles[z] and
                    self.candidate_LC['LC_Descriptor'][k] in self.triangles[z]):
50                new_triangle = False
51              if(new_triangle == True and new_triangle_i == True):
52               self.candidate_LC['LC_Security'][i] += 1
53              if(new_triangle == True and new_triangle_j == True):
54               self.candidate_LC['LC_Security'][j] += 1
55              if(new_triangle == True and new_triangle_k == True):
56               self.candidate_LC['LC_Security'][k] += 1
57              # Add the new triangle (ID_S + Descriptor of the 3 participants)
58              if(new_triangle == True):
```

```
59              self.triangles.append([self.candidate_LC['LC_ID_S'][i],
                    self.candidate_LC['LC_ID_S'][j], self.candidate_LC['LC_ID_S'][k],
                    self.candidate_LC['LC_Descriptor'][i],
                    self.candidate_LC['LC_Descriptor'][j],
                    self.candidate_LC['LC_Descriptor'][k]])
60              self.reputation[str(self.candidate_LC['LC_ID_S'][i])] +=
                    (int(self.candidate_LC['LC_Security'][i]))
61              self.reputation[str(self.candidate_LC['LC_ID_S'][j])] +=
                    (int(self.candidate_LC['LC_Security'][j]))
62              self.reputation[str(self.candidate_LC['LC_ID_S'][k])] +=
                    (int(self.candidate_LC['LC_Security'][k]))
63            new_triangle = True
64            new_triangle_i = True
65            new_triangle_j = True
66            new_triangle_k = True
67            # Send back the validated LCs, if not already published, the field
                  'LC_Descriptor' univocally defines
68            if(self.candidate_LC['LC_Security'][i] >= security_level):
69             if(self.candidate_LC['LC_Descriptor'][i] not in
                    self.validated_LC['Descriptor']):
70              self.validated_LC['ID_Sender'].append (self.candidate_LC['LC_ID_S'][i])
71              self.validated_LC['Descriptor'].append
                    (self.candidate_LC['LC_Descriptor'][i])
72              self.balances[str(self.candidate_LC['LC_ID_S'][i])] += 2
73            if(self.candidate_LC['LC_Security'][j] >= security_level):
74             if(self.candidate_LC['LC_Descriptor'][j] not in
                    self.validated_LC['Descriptor']):
75              self.validated_LC['ID_Sender'].append (self.candidate_LC['LC_ID_S'][j])
76              self.validated_LC['Descriptor'].append
                    (self.candidate_LC['LC_Descriptor'][j])
77              self.balances[str(self.candidate_LC['LC_ID_S'][j])] += 2
78            if(self.candidate_LC['LC_Security'][k] >= security_level):
79             if(self.candidate_LC['LC_Descriptor'][k] not in
                    self.validated_LC['Descriptor']):
80              self.validated_LC['ID_Sender'].append (self.candidate_LC['LC_ID_S'][k])
81              self.validated_LC['Descriptor'].append
                    (self.candidate_LC['LC_Descriptor'][k])
82              self.balances[str(self.candidate_LC['LC_ID_S'][k])] += 2
```

B | Additional examples of Byzantine robots effects

Here, there are reported additional visual examples to better understand the advantage of having a Byzantine-fault tolerant system.
In particular, the maps where the effect of many Byzantine robots in action destroys the localisation procedure are illustrated and compared to the map that is typically found when the system is protected with my protocol. This chapter complements the results showed in 4.2.

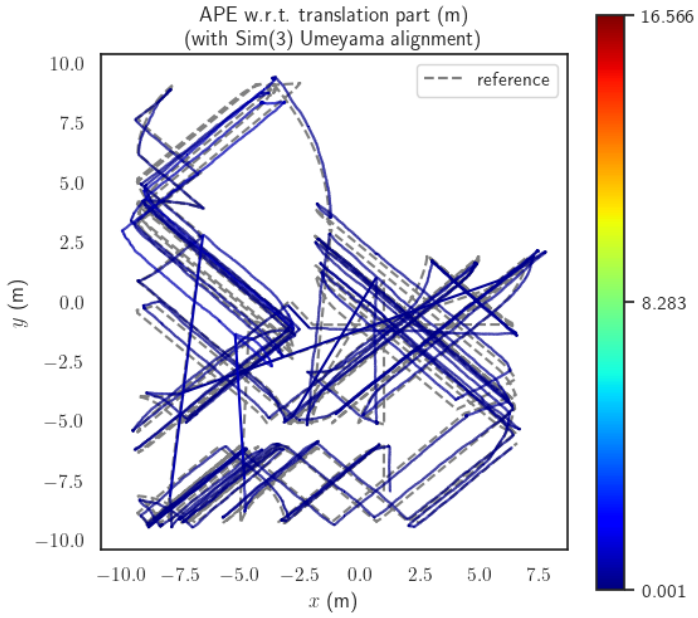

Figure B.1: Secured Swarm-SLAM. 1 Byzantine robot.

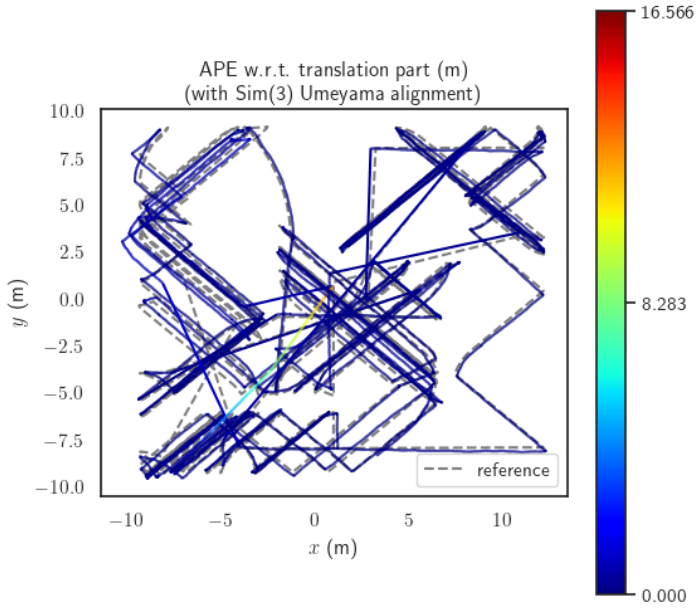

Figure B.2: Non secured Swarm-SLAM. 1 Byzantine robot.

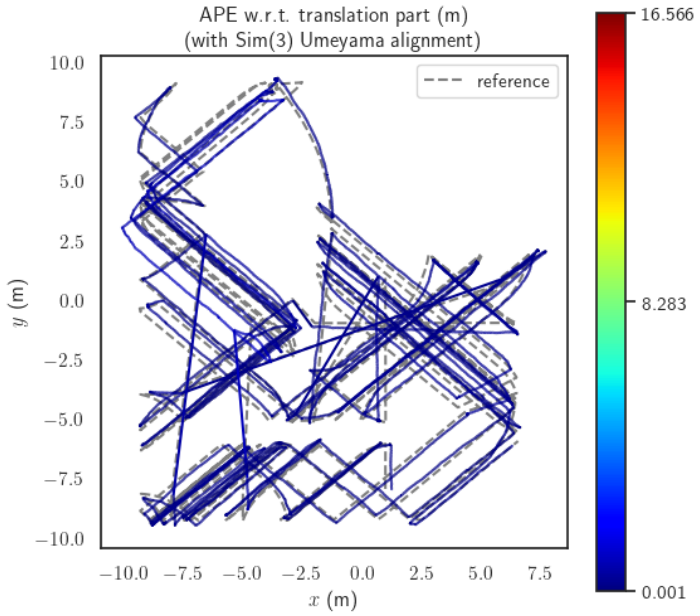

Figure B.3: Secured Swarm-SLAM. 2 Byzantine robots.

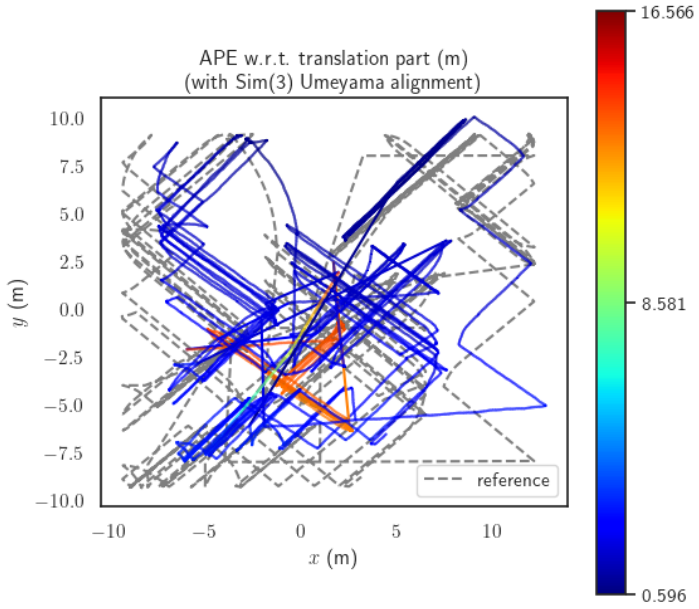

Figure B.4: Non secured Swarm-SLAM. 2 Byzantine robots.

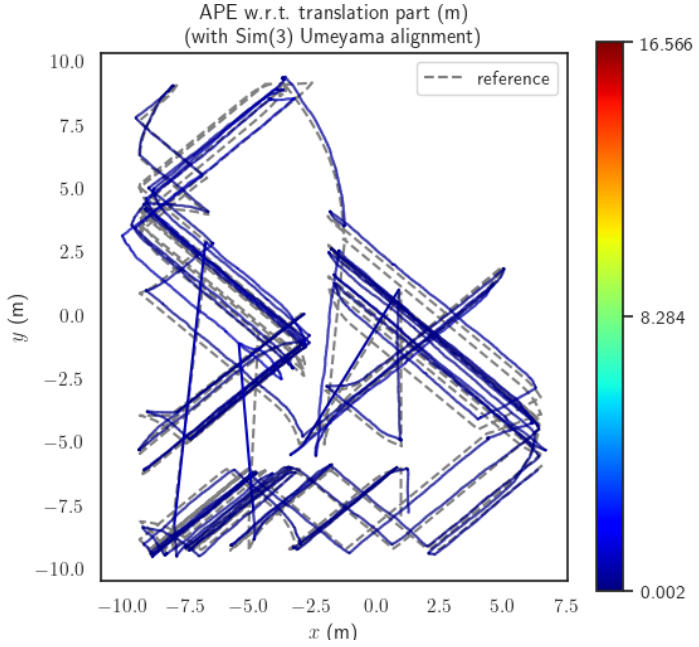

Figure B.5: Secured Swarm-SLAM. 3 Byzantine robots.

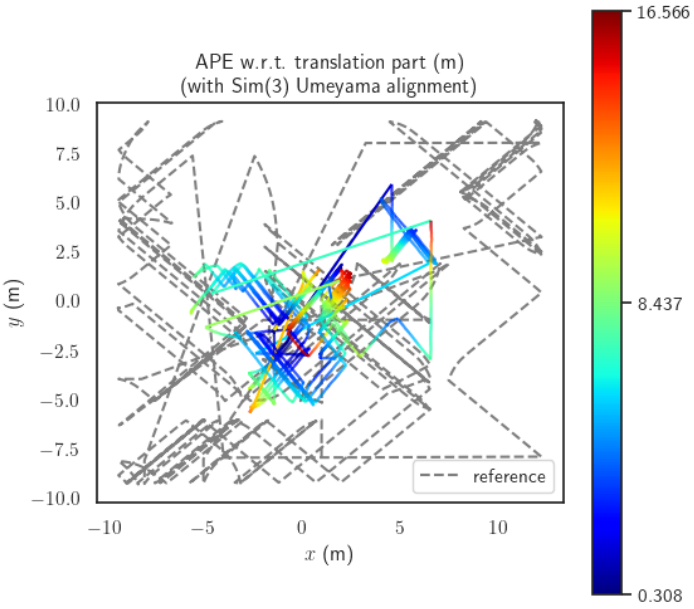

Figure B.6: Non secured Swarm-SLAM. 3 Byzantine robots.

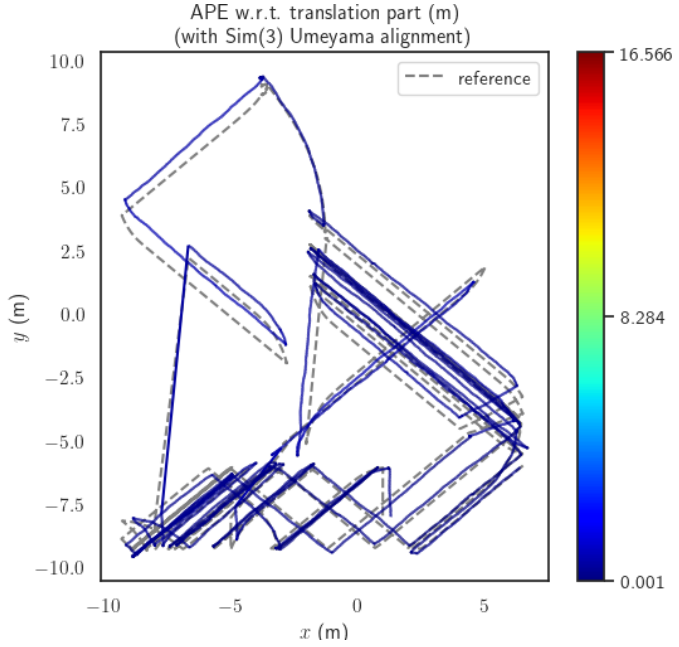

Figure B.7: Secured Swarm-SLAM. 4 Byzantine robots.

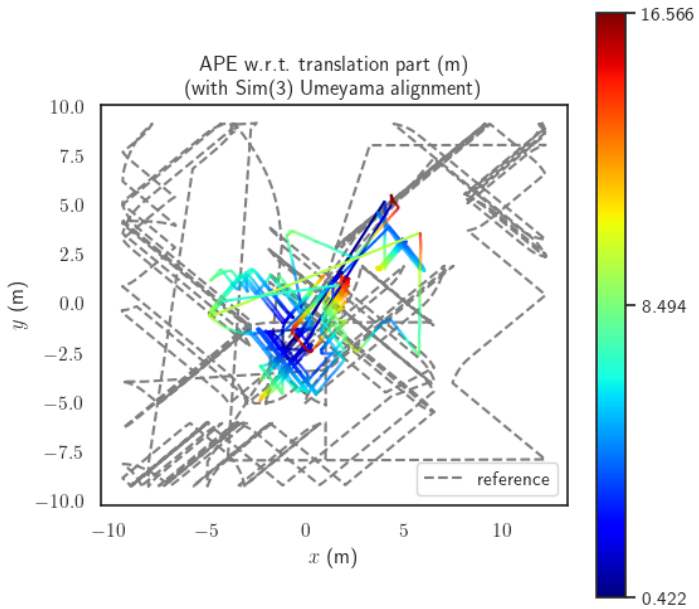

Figure B.8: Non secured Swarm-SLAM. 4 Byzantine robots.

List of Figures

List of Tables