ECOLE
POLYTECHNIQUE
DE BRUXELLES

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

An Information Market for Social Navigation in Robots

Mémoire présenté en vue de l'obtention du diplôme
d'Ingénieur Civil en informatique à finalité spécialisée

Ludéric Van Calck

Directeur
Professeur Marco Dorigo

Superviseur
Dr Andreagiovanni Reina

Co-Superviseur
Alexandre Pacheco

Service
IRIDIA

Année académique
2021 - 2022

ULB UNIVERSITÉ LIBRE DE BRUXELLES

ULB UNIVERSITÉ LIBRE DE BRUXELLES

Exemplaire à apposer sur le mémoire ou travail de fin d'études,
au verso de la première page de couverture.

Réservé au secrétariat :	**Mémoire réussi***	**OUI**
		NON

CONSULTATION DU MEMOIRE/TRAVAIL DE FIN D'ETUDES

Je soussigné

NOM :
VAN CALCK
……

PRENOM :
Ludéric
……..

TITRE du travail :
An Information Market for Social Navigation in Robots
……

……
……

AUTORISE*

~~**REFUSE***~~

la consultation du présent mémoire/travail de fin d'études par les utilisateurs des bibliothèques de l'Université libre de Bruxelles.

Si la consultation est autorisée, le soussigné concède par la présente à l'Université libre de Bruxelles, pour toute la durée légale de protection de l'œuvre, une licence gratuite et non exclusive de reproduction et de communication au public de son œuvre précisée ci-dessus, sur supports graphiques ou électroniques, afin d'en permettre la consultation par les utilisateurs des bibliothèques de l'ULB et d'autres institutions dans les limites du prêt inter-bibliothèques.

Fait en deux exemplaires, **Bruxelles, le** 03/06/2022

Signature

Van Calck

Formulaire version 04/2015

** Biffer la mention inutile*

** Biffer la mention inutile*

Abstract

Dans le domaine de la robotique en essaim, on considère généralement que des robots coopératifs, malgré leurs capacités individuelles limitées, peuvent résoudre des problèmes complexes en travaillant ensemble. Cependant, dans le contexte d'un essaim libre, où des robots différents peuvent être ajoutés par des acteurs aux intérêts potentiellement concurrents, la coopération apparaît comme une stratégie parmi d'autres. Par conséquent, nous envisageons dans ce rapport un marché d'échange d'informations, où des robots peuvent acheter et vendre des informations à d'autres robots via des transactions enregistrées et sécurisées par la blockchain, un système de paiement distribué, et où la coopération et l'honnêteté sont encouragées par les mécanismes économiques sous-jacents. Pour démontrer l'intérêt d'un tel dispositif, nous élaborons un simulateur informatique, où des robots virtuels doivent chercher de la nourriture concentrée en un endroit et la ramener à leur nid. Leur coopération, en achetant et vendant des informations à d'autres robots, est un ingrédient indispensable afin d'effectuer cette tâche de manière efficace. Par ailleurs, nous illustrons comment l'insertion d'un seul Byzantin, qui ment à propos de l'information qu'il vend, est capable de lourdement perturber le système, et de compromettre l'efficacité de l'ensemble du groupe. Dans cette optique, nous introduisons deux mécanismes de protection. D'une part, en rendant les robots honnêtes plus sceptiques, nous montrons qu'ils sont capables de détecter et d'ignorer les fausses informations, au prix d'une efficacité plus faible dans un groupe complètement coopératif. D'autre part, via l'instauration de principes économiques intelligents, nous montrons comment, dans un essaim composé en majorité de robots honnêtes, vendre des informations authentiques est plus rentable que de vendre de fausses informations. Ainsi, les robots honnêtes accumulent plus d'argent que les Byzantins, et nous postulons que cette différence de richesse peut ensuite être exploitée pour ignorer, voire excommunier, les robots Byzantins, et limiter leur influence sur l'efficacité de l'essaim. Cette étude est la première à utiliser des mécanismes économiques pour encourager les comportements coopératifs dans des essaims de robots. Nous pensons que ce travail peut ouvrir la voie à de la recherche future dans l'économie appliquée à la robotique en essaim, en exploitant l'opportunité qu'offre la technologie de la blockchain dans la conception de systèmes économiques décentralisés.

Abstract

Robot swarms are generally considered to be composed of cooperative agents that, despite their limited individual capabilities, can achieve complex tasks by working together. However, in contexts such as an open swarm, where different robots can be added to the swarm by different parties with potentially competing interests, cooperation is but one of many strategies. As such, we envision an information market where robots can buy and sell information through transactions stored on a distributed blockchain, and where cooperation and honesty are encouraged by the economy itself. As a proof of concept, we design and build a multi-agent simulator where robots perform a classical foraging task, and where cooperating by buying and selling information with other robots is paramount to accomplish the task efficiently. We illustrate further that including even a single Byzantine robot that lies to others can heavily disrupt the system and compromise collective efficiency. Hence, we devise two protection mechanisms. On the one hand, through increased skepticism, robots can detect and discard Byzantine information, at the cost of a lower efficiency in a fully cooperative swarm. On the other hand, through smart economic rules, we show how in a swarm primarily composed of honest robots, selling honest information is more profitable than selling false information, leading to honest robots acquiring more wealth than Byzantines. We suggest this discrepancy on wealth can then be used by the swarm to ignore or blacklist Byzantine robots and limit their negative impact on the swarm's efficiency. This is the first study that employs economic mechanisms to encourage cooperative behavior in robot swarms. We believe that this thesis can pave the way for further research in economic-based swarm robotics exploiting the timely opportunity for decentralized economies offered by blockchain technology.

Acknowledgements

I would first like to thank my supervisor Andreagiovanni Reina and co-supervisor Alexandre Pacheco for their continuous help, implication, and support throughout the whole year. They gave me the freedom to be creative and explore the ideas I found most interesting, while providing sufficient direction to build a stable foundation to my work, and to ensure its steady growth and improvement. I am ever grateful for their frequent warm encouragements, thoughtful advice, and helpful criticism. I can confidently say they are both a big part of why I am proud of the work I achieved, and why I will keep a fond memory of this journey.

I would further like to thank Professor Marco Dorigo, and Volker Strobel for their insight and constructive comments that helped me tie the loose ends, as well as prepare the upcoming oral defense, of this thesis.

I would like to acknowledge my reviewer Raina Zakir and wish her a pleasant (and hopefully interesting) read.

Finally, I want to express my gratitude to my family and friends, who provided me with the perfect setting to succeed.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Drawing inspiration from nature and its flocks of birds or social insects, swarm robotics deals with the design of systems where large groups of simple robots can collectively achieve a task or solve a problem. In order to do so, they need to cooperate, but due to environmental and/or hardware constraints, this can only happen through local interactions and communication. This means there is no central authority dictating the robots how to act but they are rather able to self-organize by observing their surroundings (other robots or the environment itself) and following simple rules according to the information they gather.

It is commonly accepted that robots are cooperative by nature, meaning that they openly communicate with other robots and do not deliberately lie when sharing information, since they are assumed to all be part of the same swarm sharing a common goal. In this thesis we consider the new concept of an *open swarm*, where robots may be added to the swarm by different, possibly competing, parties (e.g. different companies interested in solving the same problem). Consequently, cooperation can be seen as one strategy among others, whose main benefit comes from the swarm's emergent self-organization. In this context, we introduce the notion of an information market, a framework regulating how robots may buy and sell information with their peers, where our goal is to provide economic incentive for robots to be honest and cooperative. To do so, we specifically study the impact of Byzantine robots (i.e. robots undermining the swarm's operation, either due to faultiness or malicious intent) on the swarm's efficiency, and present the information market as a tool to protect honest robots from being disrupted.

1.2 Research Objective

The goal of this thesis is to provide a proof of concept for an information market in the context of a social navigation task (more precisely: central place foraging), through the elaboration of a multi-agent Python simulator. In our model, robots are represented as point-like particles in a 2D plane, that keep track of their own movement through odometric estimates and are deprived of any GPS or similar tools. Since these estimates are subject to noise, robots frequently get lost. To find their goal they have the choice between exploring the environment, or buying information on where they need to go from other robots. We assume robots use a blockchain and execute smart contracts to respectively record transactions and calculate information prices, in a secure and distributed

manner. Since the technical implementation of a distributed blockchain with smart contracts is outside the scope of this work, we implement a simplified centralized system that emulates its properties.

Furthermore, to illustrate this proof of concept's interest and usefulness, we use it to study the effect of Byzantine robots selling false information to honest robots. We first show how Byzantines can disrupt a swarm composed of simple naive robots. Secondly, we introduce individual protection mechanisms, where robots can detect false information by comparing information from multiple sources, to increase the swarm's resiliency. Finally, we explore ways to use the market to penalize false information with the ultimate goal of using robots' wealth as an indication of their trustworthiness.

1.3 Outline of the Thesis

The thesis is divided into three parts. In the first part we analyse state of the art scientific literature for collective foraging, blockchain-based robot swarms and the resiliency of robot swarms facing faulty or malicious agents. In the second part, we describe our model for social navigation, robot behaviors, individual protection measures (by comparing information from multiple sources), systemic protection measures (through the use of the information market), and their technical implementation. In the third and final part, we present and discuss our results pertaining to the individual and systemic protection measures described in the second part, before ending with a description of what remains to investigate in future work, and a conclusion.

Part I

State of the Art

Chapter 2

Collective Foraging

In biology, foraging is the action of searching for resources, such as food, in the environment [5]. *Collective* foraging is thus the activity where a group of organisms collectively searches for these resources in their environment. When resources are grouped in a central location and have to be brought back to another central location (e.g. ants traveling back and forth between some food and the colony), the field of study is known as central place foraging [22].

2.1 Indirect Communication or Stigmergy

Many species of ants and other social insects tend to use an indirect form of communication, known as stigmergy, to signal they have found a food source [40]. Ants coming back from the food source modify the environment by leaving in their trail a chemical trace, or pheromone, that other ants can detect. Ants then tend to favour following high-concentration pheromone trails, creating a positive feedback loop [33]. Using this mechanism, ants are able to solve different types of problems. For example, by modulating the pheromone quantity deposited based on the food source quality, ants are able to choose the best food source in the area [3, 28]. They can also converge on the shortest path from the food source to the nest, since for a same number of ants, shorter paths will accumulate pheromone more rapidly than longer paths [15].

Drawing inspiration from these natural phenomena, stigmergy has also been proposed as a method of indirect communication to solve foraging tasks in robot swarms. However, the artificial replacement to chemical pheromones ants use is not straightforward. Some research provides physical means to modify the robot's environment analogously to pheromones, such as phosphorescent paint [20], or ethanol [14]. Another approach is to use a "smart" environment that can store virtual pheromone information and communicate it with robots. Even though this would be difficult to implement in real-life applications, it is a very popular approach to conduct research in the lab. Implementations of such smart environments are for example based on radio-frequency identification technology (RFID) [19], augmented reality (AR) [29], or other specialized hardware [37].

Regardless of its practical implementation, multiple studies have found that stigmergy is an effective mechanism to reproduce the swarm organization observed in ants, to effectively navigate between food and nest in a central place foraging context [20, 35, 18].

2.2 Direct Communication

Since using stigmergy in a robot swarm may not always be feasible due to environmental or hardware constraints, some studies also propose direct communication as a means of organization. In practice, robots can exchange messages or visual cues (such as through the use of LEDs) to communicate information to other robots, to try to form a chain between the food source and nest [7, 12, 16, 30, 13].

If the chain is static [7, 12, 16], the robots forming the chain, also known as beacons, do not move and guide other mobile robots efficiently between the food and the nest. The main drawbacks of this approach is that the beacons do not actively contribute to the foraging task, may need to be very numerous in large environments, and may also restrict the movement of other robots.

To mitigate these drawbacks, alternatives have been proposed where the chain is dynamic, and robots in the chain move between food and nest sites, both actively performing the task (i.e. foraging) and guiding other robots [30, 13]. For example, in a first study [30], robots simply signal their position with a blue LED other robots can detect. By moving in circular paths (all clockwise), being attracted to the food or nest site when in detection range, but dodging other robots by rotating counter-clockwise, the robots are able to straighten their initial circular motion into a straight dynamic chain between food and nest. In a second study [13], robots use a direct communication system to share information about the last time they encountered a location. By moving towards robots that saw the location more recently, the swam is also able to form an efficient dynamic chain between food an nest, both in environments with and without obstacles.

The result of these studies shows direct communication can also be used for robot swarms to self-organize and efficiently accomplish a central place foraging task.

2.3 Random Exploration

In Sections 2.1 and 2.2, we present two ways a robot swarm can efficiently navigate between two locations, and how foraging robots can converge on the shortest path. However, all of the proposed approaches rely on robots at first being able to find the locations by exploring the environment.

The efficient exploration of an environment in search of a specific location is a complex and extensively researched problem, both in biology and swarm robotics [9, 11]. Indeed, animal search patterns, can often be described by different types of random walks [2, 4, 9, 39]. In swarm robotics, the interest in random walks stems from their simplicity and versatility (since they do not require much sensing capabilities or computing power), and some specific types of random walks can be very effective in exploring unknown environments [9, 11].

Lévy walks, which model the patterns exhibited by albatrosses or deer for example [39], are often used when the search targets are sparse, as in this case, research suggests that the Lévy walk can lead to optimal searches [38]. However, this assumption should be taken with a grain of salt as other research finds this superiority to be questioned under specific conditions [24]. A Lévy walk is characterised by a distribution of time between consecutive turns, or step-length, following a power law, which leads to series of quick turns (local exploration) followed by long straight displacements (relocation). Brownian motion, which models the motion of particles colliding with molecules in a fluid for example, can be seen as a Lévy walk with a negative exponent equal to three in the power law [11].

Correlated random walks are characterised by a correlation between the successive turning angles of a moving agent, known as persistence [25], leading to the walker tending to point in the same direction for successive steps. The correlated random walk is a natural way to model animal paths in the scientific literature as they tend to move forward [36].

Finally, it is also possible to consider hybrid random walks, where the step-length follows the Lévy walk's power law, but the successive turning angles are persistent like in a correlated random walk. In a study, some researchers propose how to implement such a hybrid random walk in a robot swarm, and explore different parameter values to optimize search efficiency in various settings [11].

Chapter 3

Blockchain-based Robot Swarms

One of the main characteristics of a robot swarm is the absence of any centralized control. Rather, the swarm is able to perform complex tasks through emergent behaviors which stem from individual actions, governed by local sensing and communication capabilities. This decentralized architecture is precisely what enables a robot swarm's parallel execution, fault tolerance, and scalability features, but it comes at the cost of not disposing of a centralized system to store shared knowledge robots acquire over time (i.e., a global information database). Consequently, new research has been presented to use blockchain technology both as a shared database, as well as a way to secure a robot swarms against Byzantines (i.e. defective or malicious robots undermining the swarm's expected behavior) in the context of consensus achievement [32, 31, 23].

3.1 Blockchain Technology

Blockchain technology was first introduced in 2008 as a way to implement the Bitcoin cryptocurrency [21], where the blockchain can be seen as a common, decentralized ledger of peer-to-peer transactions. This was made possible by leveraging cryptography and the concept of Proof of Work consensus to enable network participants to update and agree on the current state of the ledger in a fully decentralized manner. More recently, the Ethereum Foundation [6] has proposed a framework to, not only store data, but also execute computer programs, known as smart contracts, through the blockchain. In the context of cryptocurrencies, smart contracts can be used to trigger automatic (i.e. programmed) financial transactions, but they can also be generalized as a means to execute any programmable code, which is itself stored on the blockchain. The blockchain can thus be seen as a secure decentralized server capable of storing data and executing code.

3.2 Application of Blockchain in Robot Swarms

Recent research has explored the idea of using blockchain technology in robot swarms. As a first proof of concept, one study devises a system to secure the swarm from Byzantines in a collective decision making problem where the swarm is tasked with determining the majority color (black or white) of floor tiles in an arena through voting [32]. The use of smart contracts allows the robots to detect and blacklist robots voting in an incoherent way. Using simulations in ARGoS [26], the researchers find that their blockchain-based approach is able to secure the swarm from attacks at the cost of a slower convergence time and increased data storing and communication requirements.

This approach is further explored by the same researchers in a sequel to this first proof-of-concept study [31]. In this second study, the robots are tasked with estimating the frequency of the colored tiles instead of deciding the majority color, are able to decide when to stop and "deliver their answer", and provide a mechanism to protect the swarm from Sybil attacks (i.e. a specific type of attack where Byzantine robots forge many identities to flood the swarm with false information). Through the use of simulations, the researchers determine the efficacy of this blockchain-based approach in this more complex context, and showcase the potential of blockchains in securing robot swarms.

Finally, a third study implements the same collective decision making task, where robots have to estimate the frequency of colored tiles, in a physical robot swarm [23]. To do so, the researchers use a mobile ad-hoc network protocol to share data between robots [17], as well as the Proof of Authority [34] concept to secure the blockchain, instead of the more computationally intensive traditional Proof of Work. In this way, they are able to show the viability of the blockchain approach to secure physical robot swarms, outside of simulation.

Chapter 4

Resiliency in Robot Swarms

As a distributed system, a robot swarm relies on the cooperation between its agents to accomplish a task, which means it is vulnerable to purposefully disruptive agents. In this context, we can analyze the impact of these disruptive agents and denote by resiliency, the swarm's capacity to limit their impact. Though research in this field is recent and developments are still ongoing, in this chapter, we analyze the resiliency of robot swarms in different contexts.

4.1 In Collective Decision-Making

In collective decision making, a robot swarm is tasked with agreeing on a decision based solely on local perception and interactions.

In a first study [10], robots are tasked with solving a best-of-n problem, where they have to choose the best out of n different options ($n = 2$ for the study). Different options are associated with different qualities, and the goal is for the swarm to agree on choosing the option with the highest quality. To complete the task, robots can both measure the quality of different options directly as well as advertise an option, in the aim of recruiting other robots to adopt the same option. The researchers specifically study the impact of zealots (stubborn robots that only advertise a single option and never change opinion) on the swarm's ability to converge towards the best option. More specifically, they study the case where zealots associated to the worst option are more numerous, and discuss under what conditions the system is still able to converge to the best option. They conclude that for the models they consider, only systems where honest robots can advertise their opinion for an amount of time proportional to the quality of their opinion, are resilient (up to a certain point) when more zealots advertise the worst option than the best.

In a second study [8], a robot swarm also tasked with solving a best-of-n problem is faced with three types of attacks. Contrarian robots always oppose the majority of the group to slow down the decision process. Wishy-washy robots change their opinion constantly and randomly, leading to the swarm not being able to stick to a decision. Groups of zealots always advertise the same inferior option to make the swarm choose an inferior-quality option. The study analyzes different decision models and their resiliency to each type of attack.

A third study [27] describes communication manipulation as an attack where a threat can manipulate, and modify, data sent between honest robots. To explore this scenario, the study analyzes the case where a few honest robots (called discoverers) from a swarm need to propagate information to the remaining robots (called receivers), while attackers try to propagate the wrong piece of information. This task is modelled as a best-of-n

problem ($n = 2$), where both pieces of information have the same quality, and where only receivers can change their opinion. The researchers propose and analyze two defense mechanisms, and show that one of these can substantially increase the swarm's resiliency, even for a number of attackers close to the number of discoverers.

Finally, we can also cite the blockchain-based robot swarm studies described in Section 3.2 [32, 31, 23] that use blockchain technology to secure a robot swarm and increase its resiliency in a collective decision-making context.

4.2 In Pheromone-Based Foraging

Robot swarms relying on stigmergic communication methods for their organization can also be attacked in ways analogous to how ants can get trapped in a pheromone loop, or "ant-mill", and die of exhaustion. In [1], the attack is analyzed using a simulator where robots are agents that solve the classical central place foraging task through the laying of pheromone. Here, a small number of attackers (relatively to the amount of honest robots) constantly lay "food" pheromone (i.e. pheromone used to indicate a trail to food) near the nest, which, once a certain threshold is reached, traps honest robots inside due to the high pheromone concentration. To mitigate this effect, honest robots can lay another type of "cautionary" pheromone when they are following the "food" pheromone. The quantity of "cautionary" pheromone deposited increases the longer a robot is following a "food" pheromone trail. When the concentration of "cautionary" pheromone exceeds that of "food" pheromone, robots stop taking the "food" pheromone into account for their navigation. In the end, this defense mechanism is shown to be an effective way to thwart the attackers' trap.

Part II

Materials and Methods

Chapter 5

Simulation Environment and Agents

5.1 Environment

The robots move in a finite 2D rectangular arena of size $W \times H$. In our study we set $W = 1200$ and $H = 600$. This space is empty except for two specific sites, the food patch and the nest, respectively located at $P_F = (200, 300)$ and $P_N = (1000, 300)$, as to be placed symmetrically to the left and right of the environment. These sites are circular areas of a fixed radius r_F and r_N, that we set to $r_F = r_N = 50$. For a graphical representation of the environment, see Fig. 5.1.

The robots have to transport items, which we allude to as "strawberries" due to their graphical depiction in the simulator, from the food site to the nest site. To simulate the time necessary for a robot to manipulate an item (i.e. the time that would be necessary to physically collect or deposit the item), the collection and deposit of a strawberry is not instantaneous when a robot enters a site. Instead, the robot needs to spend a random amount of time within the site in order to collect/deposit it. We implement this random manipulation time by placing the strawberry or deposit spot at a random point, chosen uniformly within the site, where the robot needs to travel to complete the collection or deposit action. Furthermore, once the robots are organized in an efficient path between the two sites (see section 6), this process serves to shuffle the robots' relative placement within the chain across different round-trips, making the chain more homogeneous, and preventing robots gaining any advantage from their relative position in the group when buying and selling information.

5.2 Robots

Robots are point-like particles that do not collide with each other and that can communicate when within a certain communication range $r_C = 50$. They can move freely in the environment, up to a certain maximum velocity $V = 2.5$ (measured in distance moved per simulation step) and can change direction instantly (no inertia). The robots do not have access to any global tool (e.g. a GPS) to accurately know their position and thus need to keep track of their own relative movement through odometry. However we assume odometric measurements are noisy, and accumulation of odometric noise leads robots to drift away from the trajectory they think to be following.

To model this *drift*, each robot has 2 intrinsic noise parameters: bias μ and standard deviation σ. At each step of the simulation, a robot sets its desired movement as a vector in its own reference frame. However, its actual movement is rotated by an angle sampled from a normal distribution with mean μ and standard deviation σ. We assume the

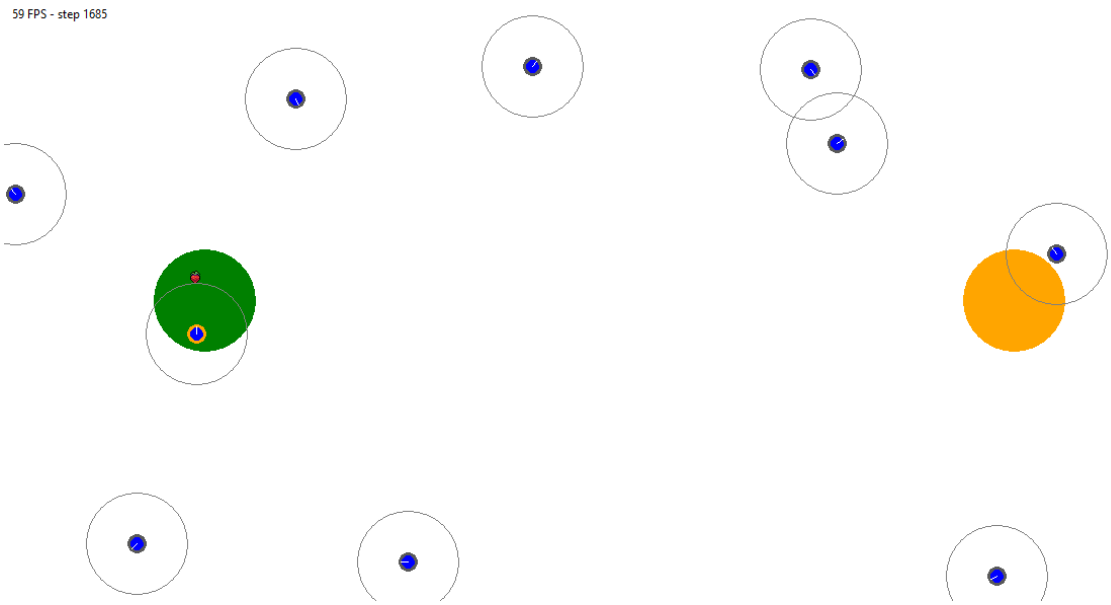

Figure 5.1: Environment containing 10 robots (small blue circles). The green circle represents the food patch, the yellow circle represents the nest, and the gray rings around the robots represent their communication range

robot uses odometry readings that indicated desired movement was carried out perfectly, therefore there is a discrepancy between the robot's recorded movement and its actual motion. As a result, a robot that thinks to move perfectly straight will actually be moving on a curved trajectory, as it is on average turning μ degrees each step.

At the beginning of each experiment, each robot samples its drift bias μ from a bimodal probability distribution $p(m_\mu, s_\mu)$ of the following form:

$$p(m_\mu, s_\mu) = \frac{1}{2}(N(m_\mu, s_\mu) + N(-m_\mu, s_\mu)) \tag{5.1}$$

where $N(m, s)$ denotes a normal distribution of mean m and standard deviation s. This means certain robots intrinsically drift more than others. Unlike μ, the drift's standard deviation σ has the same value for all robots.

Chapter 6

Social Navigation

The robots are tasked with going back and forth between a location containing food and their nest. However, due to local sensing and communication constraints, as well as noisy odometric estimates, they need to cooperate by sharing information on where they think these places are in order to efficiently navigate between them. In this chapter, we describe the base robot behavior that can store and share information to eventually form a dynamic chain, where the system converges on a single path between the food and the nest. This behavior is inspired from an article [13] we describe in Section 2.2.

6.1 Storing Information

When the robots move, they can use sensors to estimate the distance they have traveled and the angle they have turned between consecutive time steps (i.e. odometry). By continuously keeping track of these estimates, the robots are able to memorize their relative distance and orientation from a given starting position. In practice, this information is stored in a navigation table of the following form:

Location	**Distance**	**Age**	**Known**
FOOD	$(200, -5)$	78	`True`
NEST	$(-1000, 35)$	400	`True`

Table 6.1: Example Navigation Table

The first column contains the name of the point of interest. The second column contains a 2D vector of the robot's approximated distance from that point, in the robot's reference frame (here, the food is estimated to be 200 units forward and 5 units to the right). The third column keeps track of the relative age of the information, and the attribute in the last column is a boolean flag that indicates whether the information is trustworthy or not. A row in the navigation table is referred to as a *target*.

At each step of the simulation, the robots increase each target's age by 1. They also update each target's distance vector with the data from their odometric estimates, by rotating each vector with the (opposite) angle they have turned, and updating their components with any translational movement. Finally, if the robot reaches the target location but its sensors do not confirm the presence of the expected site, the robot switches the "known" flag to `False`, indicating the target is not trustworthy.

6.2 Using Information

If a robot is not carrying any food, and has a known FOOD target in its navigation table, it simply tries to go towards it. If it does not know the FOOD target location, the robot explores the environment randomly (see Section 8.8.1) until either stumbling upon the food site by chance, or until acquiring information about the FOOD's location from another robot. Symmetrically, if the robot needs to carry food back to the nest, it follows the same procedure with the NEST target.

6.3 Acquiring New Information

Any given robot has two ways of acquiring new information: sensing a location directly or getting a target from another robot. When a robot senses a given site, it can directly update its navigation table (see Tab. 6.1) with a target containing the distance to that site (from its sensors), setting the age attribute to 1 and the known variable to `True`. Alternatively, robots constantly broadcast the age and location attributes of their own known targets and any other robot within range can ask to buy the full target (for the moment we consider cooperative robots, which means the transaction is free). Robots decide to acquire another robot's target if the corresponding age is lower than their own, for a given location.

First, when acquiring a target from another robot, the distance attribute needs to be rotated and modified according to the distance between the two robots, as well as their relative orientations. Once this is done, the buyer needs to decide how to update its navigation table. If the buyer's own target is not known, the buyer simply replaces its target for that location with the new one. However, if the buyer's target has a `True` known attribute, instead of simply replacing its own target with the one it bought, the buyer combines the information from the two into a new target. The resulting target's age is the average of the two age attributes and the resulting distance vector is a weighted average of the two vectors, where the distance with a lower associated age has a higher weight. More formally:

$$\vec{v}_{res} = \frac{a_{buyer}}{a_{buyer} + a_{seller}} \cdot \vec{v}_{seller} + \frac{a_{seller}}{a_{buyer} + a_{seller}} \cdot \vec{v}_{buyer} \tag{6.1}$$

$$a_{res} = \frac{a_{buyer} + a_{seller}}{2} \tag{6.2}$$

with a representing a target's age attribute and $\vec{v}$ representing its 2D distance vector. Also, since robots only buy newer information, we have $a_{buyer} > a_{seller}$. This allows a robot to update its own target, attributing more importance to new information, without completely discarding its previous beliefs.

This reasoning relies on one main hypothesis: a target's age measures its quality. Because information is distorted through imprecise odometric estimates, older information will likely have accumulated more errors and be more imprecise. Even though this is true for information carried by a single robot, in the negotiation process, a robot needs to decide if newer information from *another* robot is better than its own. As we know, a robot's intrinsic odometric drift is a random variable (see Eq. 5.1), which means newer information from a robot with a large drift may actually be less accurate than older information from a robot with a small drift. Therefore, there could be value in robots estimating their own drift, to more precisely assess the quality of their own information. To test whether robots having an intrinsic estimate of their own drift is worth it, we conducted experiments where

a target's age attribute was replaced with a *quality* attribute, varying between 100% and 0%. At each step of the simulation, the robots would decrease this quality attribute (analogously to increasing an information's age), and would use it to determine whether they should buy another robot's target.

In a first set of experiments, all robots decrease this quality attribute by the same constant amount at each step of the simulation, which results in this quality attribute being completely equivalent to age. In a second set of experiments, robots decrease their quality attributes by an amount proportional to their drift (we assume they have access to its value, even though in practice, they would need to estimate it), which means robots drifting more degrade the quality faster. The third and final set of experiments is similar to the second, except the quality of information decreases exponentially with a robot's drift (penalizing heavy drifters more).

Since the results of these experiments are similar, we conclude that estimating a robot's intrinsic drift, which is necessary to use the quality measure used for the second and third set of experiments, (which would not be as accurate as for these control experiments where we gave perfect information about the magnitude of their drifts to the robots) wouldn't be worth it compared to simply using a target's age as a measure of its quality.

Chapter 7

Dealing with Misinformation

In the previous chapter we described how basic honest cooperative robots can freely exchange information to efficiently complete the foraging task. In this chapter, we study what happens when we introduce liars (known as Byzantine robots in swarm robotics terminology), i.e. robots that modify the targets they give to other robots, resulting in a destabilized system.

7.1 Introducing Saboteurs

To study the impact of Byzantine robots, let's introduce a new kind of behavior that lies about the information it communicates to other robots: the saboteur behavior. Saboteurs behave in a completely identical way to the basic behavior described in Section 6 except for a single difference: when selling a target, they rotate its distance attribute vector by 90 degrees. This means that instead of being redirected to the desired location, robots acquiring this target are sent off in a completely different direction. Unfortunately a single saboteur can completely disorganize the entire system, and break any dynamic chain formed by the collaborative robots. The ways to counter this Byzantine behavior can be separated into two distinct categories: individual (or micro) and systemic (or macro) protection.

On one hand, the idea behind micro-level protection is that robots try to individually filter out or lessen the impact of a Byzantine target on their own movement. We consider two of such strategies: the careful and smart behaviors, in Section 7.2.

On the other hand, macro-level protection comes from designing a set of rules all robots must follow in such a way that honest robots can be distinguished from Byzantines. To this end, we introduce different payment systems (regulating the price of buying and selling information) and reward mechanisms (regulating the reward for completing the task) with the intention of finding a set of rules encouraging honesty and cooperation by attributing a higher proportion of the total wealth to honest robots compared to Byzantines (see Section 7.3).

7.2 Individual Protection

7.2.1 Careful Behavior

Instead of comparing and combining new targets one by one, careful robots wait until having acquired a certain number of targets before changing their beliefs. This procedure can be seen as collecting information from multiple sources before making a decision. In

practice, careful robots decide to buy targets in the same way as basic robots (decide to acquire target if its age is lower), but instead of directly combining this new target with their own, they store it in a *pending information table*. This table is very similar to the robot's navigation table, as its targets are updated at each step according to odometric estimates, except it is not used to dictate how the robot should move.

Once a certain number s of targets have accumulated for a given location in the pending information table, the robot combines the acquired information with its own target. To combine this information, the robot first computes the average of the s pending targets' distance vectors. The target whose vector is closest to this average is then chosen as a replacement for the target in the robot's navigation table, while the others are discarded. The s parameter is called the *security level* and its value can be tuned to set (as the name says) the desired level of security. A security level that is too low increases the risk that the robot accepts an outlier, while a security level that is too high introduces high latency before employing acquired information (in turn lowering the benefits of frequent cooperation).

7.2.2 Smart Behavior

Whereas the careful behavior accumulates a fixed number of targets before choosing one that may be arbitrarily different from the robot's current belief, smart robots accumulate an arbitrary number of targets in their pending information table until any pair (either between a pending target and the robot's current belief, or between two pending targets) are similar enough.

In a nutshell, whenever a smart robot obtains a new target, it compares it with the one in its navigation table as well as those in its pending information table by computing a *difference score* for each pair. If at any point, this difference score is lower than a threshold θ, the new target replaces the one in the robot's navigation table. This difference score is a measure of how different two targets are.

In practice the difference score is computed as follows:

$$\text{diff}(i, j) = \frac{||v_i - v_j||}{||v_i||} \tag{7.1}$$

where i is the "old" target, j is the new target being compared to others, v_i designates target i's 2D distance vector attribute, and $||...||$ denotes the Euclidean norm of a 2D vector.

7.3 Systemic Protection

The goal of this section is to introduce system-wide rules or protocols regulating the price of buying and selling information, as well as the price a robot is rewarded for transporting an item to the nest. Indeed, when a robot deposits an item in the nest, it obtains a monetary reward R. This money it accumulates can, in turn, also be used to buy information from other robots.

7.3.1 Information Payment Systems

Fixed Price Transactions

The most basic information payment scheme is a buyer paying a fixed price in exchange for a seller's target. In this case, sharing information is incentivized as the the seller

obtains an immediate reward. We state this payment system here due to its simplicity, but we do not actively use it in the rest of our analysis.

Reward Share Transactions

Here, instead of paying a seller immediately, the transaction is stored until the buyer has transported an item to the nest. In the Reward Share Transaction system, when the robot deposits an item and obtains the reward R, it keeps a fraction a of this reward, while the remaining $(1-a)$ is distributed evenly between all previously stored transactions' sellers. More formally the robots are rewarded as follows:

- $W = a \cdot R$, is the worker's reward (robot transporting the item)

- $S_t = \frac{1-a}{|T|} \cdot R$ (for $t = 1, ..., |T|$) is the reward for the seller involved in transaction t in the set T of all transactions recorded during the worker's last round-trip

Window-Filter Reward Share Transactions

This payment scheme is based on the Reward Share Transactions mechanism, where instead of assigning equal weights to each transaction, similar information receives a higher share of the combined buyers' cut. The hypothesis is that in a swarm composed of a majority of honest (smart) robots, true information generally points in the same direction, and saboteur information is less common and points in another direction. By rewarding targets based on how similar they are to other targets, less numerous saboteurs should obtain a lower share than more numerous honest robots. Each transaction stores both the location of the sold target, as well as the angle the buyer should turn to follow the target's vector.

The weights are computed in the following way:

$$\text{similar}(t_i, t_j) = \begin{cases} 1, & \text{if } |o_{t_i} - o_{t_j}| < 30^\circ \text{ AND } \text{location}_{t_i} = \text{location}_{t_j} \\ 0, & \text{otherwise} \end{cases} \tag{7.2}$$

$$w_t = \sum_{i=1}^{|T|} \text{similar}(t, t_i) \tag{7.3}$$

where $\text{similar}(t_i, t_j)$ determines whether transactions i and j are considered similar (based on if the sold information pertain to the same location, and if the targets' relative orientations o to the buyer are in a 30° window), and the weight w_t of transaction t counts the number of similar transactions. One can note that $\text{similar}(t, t) = 1$, which implies $w_t \geq 1$.

7.3.2 Reward Mechanisms

Fixed Reward

For this very basic reward mechanism, a robot receives a fixed amount for bringing back food to the nest. This amount has arbitrarily been set to $R = 1$.

Round-trip Duration Reward

The Round-trip Duration Reward mechanism modifies the reward a robot gets when achieving the task based on the completion time. The reward is higher for more efficient

robots. The hypothesis is that when combining this reward mechanism with the Reward Share Transactions information payment scheme, robots are encouraged to sell accurate information, so that the buyer is more efficient in their journey, which generates a higher reward, and thus a bigger total share for both the target buyer and seller.

Practically, the reward's value is calculated as follows:

$$R(\tau) = e^{1-\frac{\tau}{\tau_{min}}} \tag{7.4}$$

$$\text{with } \tau_{min} = \frac{2D}{V} \tag{7.5}$$

where τ is the time the robot took to make a round-trip between the food and nest locations, and τ_{min} is the minimum theoretical time to travel between the two locations, calculated by dividing $2D$, twice the distance between the food and the nest, by V, the robot's maximum velocity. This means the reward's value is $R = 1$ for $t = \tau_{min}$ and follows a decreasing exponential for $\tau > \tau_{min}$.

7.4 Introducing Greedy Robots

Due to the introduction of the information payment systems, as well as the reward mechanisms, we also analyse the impact of another type of Byzantine behavior: greedy robots. Unlike saboteurs who simply disrupt the system by sending other robots off in the wrong direction, greedy robots exploit the payment system by lying on the age attribute of their targets and consistently setting it to 1. By doing so, since the decision to buy information from another robot depends on the target's age (i.e. if it is newer than what the buyer already has), greedy robots ensure they always sell their target to any robot within communication range.

Chapter 8

Technical Description

In this chapter, we describe in detail how the model described in the previous chapters is implemented. All of the code can be found on the simulator's github repository: `https://github.com/ludericv/information-market`.

8.1 General Architecture of the Simulator

The simulator is controlled by a core *MainController* class. The MainController is in charge of initializing all other software components at the start of a simulation and provides methods to retrieve data in the end (so it can be stored to a file). It uses parameters initially specified in an external configuration file (whose path is passed to the python script executing the simulation as a command line argument) and stored in a *Config* object, to start up the other classes.

The configuration file can specify to launch the simulation with visualization. In this case, the *ViewController* is initialized by the MainController to create a GUI and manage graphical components. With visualization, the MainController also provides an interface for the ViewController to request executing a new step of the simulation, when a new frame can be drawn. When visualization is turned off, the MainController never initializes the ViewController, and directly executes all the steps of the simulation in a simple loop.

With or without visualization, the *MainController* is also responsible for creating the *Environment* the simulation takes place in. The Environment class, as its name suggests, is tasked with managing the state of the environment throughout a simulation. It is responsible for initializing, storing and updating the robots, or *Agent* objects, as well as managing the Agent's interactions with the food and nest sites. The Environment class also contains a *PaymentDatabase* object, that simulates a blockchain by recording transactions between robots and managing the robots' monetary balance when buying or selling information, according to one of the payment systems described in Section 7.3.1. Finally, the Environment contains a *Market* object responsible for calculating a robot's reward when depositing an item according to one of the reward mechanisms described in Section 7.3.2.

The Agent class is responsible for managing a robot's internal state. It abstracts away the robot's hardware capabilities such as its movement and communication. This class also contains a *Behavior* object, representing the robot's control software, containing the base code for the robot's navigation (see Section 6) and possibly implementing an individual protection strategy (see Section 7.2), and/or Byzantine behavior (Sections 7.1 and 7.4). To be precise, the Agent asks its Behavior to decide what to communicate (i.e.

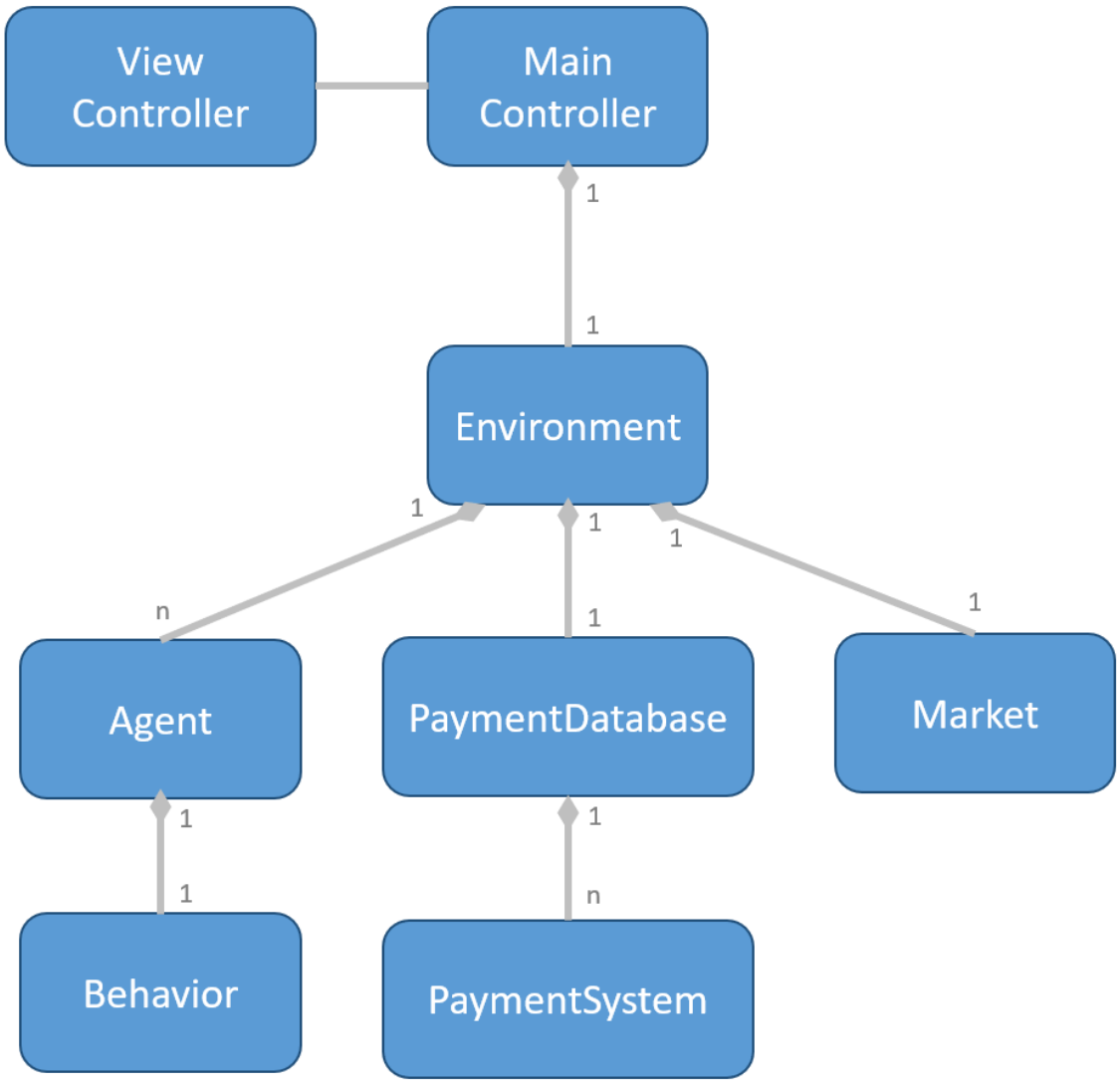

Figure 8.1: Simplified UML Diagram of the Simulator Architecture

what information to buy from other robots) and how to move, and then carries out those decisions.

A simplified UML representation of the simulator's software architecture can be found in Fig 8.1.

8.2 Synchronous Simulation

The execution of a simulation is synchronous. This means a simulation is divided in a number τ_{max} of individual time steps. At each of these steps, the Environment is updated (see Section 8.3.2), which, in turn, updates the different components it contains (i.e. robots, payment database, ...). For example, for the robots, this means moving up to a maximum distance (determined by their maximum velocity V), and buying/selling targets from other robots within their communication radius.

8.3 Environment

The Environment class models the setting the simulation takes place in. This means it manages many different objects: the robots, food and nest sites, payment database and market (i.e. the reward mechanism).

8.3.1 Initialization

The Environment is initialized by the MainController. In this step, the Environment instantiates N_r robots (i.e. Agent objects), with N_1 robots having one type of Behavior, and N_2 having another type of Behavior (such that $N_r = N_1 + N_2$). These Agents are stored in a python list known as the *population* and are attributed an ID, which is simply their index in the population list. When creating the Agents, the Environment sets their location at a random point within the arena. The Environment also creates the food and nest sites, the PaymentDatabase and Market.

8.3.2 Step

The Environment's `step` method is called by the MainController at each step of the simulation. It performs certain actions in the following order:

1. Determining which robots from the population are within communication range of each other
2. Calling every Agents' `communicate` function, where robots can communicate and buy information from other robots within communication range (in a first `for` loop)
3. Calling every Agents' `step` function, where the robots can decide their movement for that time step (in a second `for` loop)
4. Calling the PaymentDatabase and Market's `step` function

8.4 Agent

The Agent class is responsible for managing a robot's state (i.e. its position, orientation, number of items collected, etc.) and providing methods to modify this state. It also contains a Behavior object, which the Agent updates at each step of the simulation to retrieve the Behavior's decision on how to move and buy or sell information.

8.4.1 Initialization

N_r Agent objects are created by the Environment class at the beginning of a simulation and each Agent stores the parameters the Environment passes through the constructor. The Agent class randomly and uniformly samples a robot's orientation as an angle between $0°$ and $360°$, and also samples the robot's drift bias μ like described in Section 5.2.

8.4.2 Communicate

The `communicate` method is called at each simulation step by the Environment, and receives a list of references to Agents within communication range, known as neighbors, as an argument. This method performs two actions.

First it makes a copy of the Agent's Behavior's navigation table, containing the information that other Agents can access during the current communication step. This ensures the simulation remains synchronous, so Agents whose communication step happens later in the loop after the current Agent do not have access to the information gathered by the current Agent in the current time step. For example, consider three robots A, B and C, where A and B are within communication range, robot B and C are also within communication range, but A and C are not (B is in the middle of A and C). If B buys information from A, C shouldn't be able to buy that same information from B in the same time step. This information would be accessible to C in the next time step, if B and C are still within communication range.

Secondly, the `communicate` method initializes a *CommunicationSession* object, serving as an API and implementing the protocol the Behavior can use to perform transactions, to buy information from neighbors. The Behavior's `communicate` method is then called, receiving this CommunicationSession as argument.

8.4.3 Step

The Agent's `step` method is responsible for the robot's movement. First it asks the environment for sensor information. This is a dictionary of six boolean values: four proximity readings (one for each direction front, left, right and back) set to `True` if the robot is at a distance lower than its maximum velocity V from a wall in that direction, and two location readings (one for the food site and one for the nest site) set to `True` if the robot is within the site.

Second it creates an *AgentAPI* object, containing methods the Behavior object can call to get other information about the Agent (such as the maximum velocity V value, whether the robot is carrying a strawberry, or to get a random turn direction respecting the random walk pattern described in Section 8.8.1).

Finally, it calls the Behavior object's `step` function, responsible for deciding the robot's desired movement, and updating any navigation state or information. This desired movement is then retrieved as a 2D vector from the Behavior's `get_dr` function. Since this vector is expressed in the robot's local coordinate system, it needs to first be expressed in the environment's coordinate system. In 2D, this transformation is done by rotating the vector by the robot's orientation. This desired movement is then further rotated by a random angle (in degrees) sampled from a normal distribution of mean μ and standard deviation σ, like described in Section 5.2, to simulate odometric drift. The robot's position is then modified by this noisy desired movement vector (making sure the robot does not leave the environment's bounds), and the robot's orientation is changed to the direction of motion.

8.5 Behavior

The Behavior class implements the equivalent of a physical robot's control software. In its implementation, the code uses the object-oriented programming technique known as inheritance to reuse the same code for different Behavior variations. The abstract empty Behavior class specifies the basic architecture a Behavior should have through four methods that will be described in this section: `communicate, step, get_dr` and `get_target`.

The HonestBehavior class inherits from the abstract Behavior class and implements the base social navigation principles described in Section 6. The SmartBehavior and CarefulBehavior inherit from HonestBehavior and modify its `communicate` and `step` methods

to reproduce what we described in Section 7.2. The SaboteurBehavior and GreedyBehavior also inherity from HonestBehavior but change its `get_target` method to reproduce the Byzantine strategies respectively described in Sections 7.1 and 7.4. In the same way, the SmartboteurBehavior (contraction of "smart" and "saboteur") and SmartGreedyBehavior classes inherit from the SmartBehavior class, to share the individual protection measures, but replace the `get_dr` method.

8.5.1 Initialization

At initialization, a base Behavior class creates an empty *NavigationTable*, that will store the robot's estimate of the food and nest sites' locations (see Tab. 6.1 for an example), and initializes a 2D vector, $dr = (0, 0)$, representing the robot's desired movement, in the robot's reference frame. The HonestBehavior also initially sets a *state* variable to EXPLORING (which can then change to SEEKING_NEST or SEEKING_FOOD), which it uses to decide how to move. Finally, some children of the base HonestBehavior class (the SmartBehavior, CarefulBehavior classes and their children, to be precise) initialize other variables (such as the pending information table described in Section 7.2).

8.5.2 Communicate

The `communicate` method is called by the Agent class with a CommunicationSession object as an argument. Using this CommunicationSession, the Behavior can get its neighbors' metadata for both possible locations. The metadata of a location are the age and known attributes a robot broadcasts for the information it has about that location (see Tab. 8.1).

Neighbor ID	Age	Known
15	200	`True`
2	560	`True`
19	3000	`False`

Table 8.1: Example metadata for the FOOD location of a robot with three neighbors.

Based on these metadata, the Behavior can buy the full target from one of its neighbors using the `make_transaction` method from the CommunicationSession (passing it the desired location and neighbor ID), to also obtain the distance vector attribute. In this case, the CommunicationSession also notifies the Environment to record the transaction in the PaymentDatabase.

Lastly, when a transaction happens, the CommunicationSession object abstracts away the protocol needed for robots to exchange relative orientations (since targets are vectors in a given robot's own reference frame), by rotating the seller's target to the buyer's reference frame. The buyer also has access to the distance to its neighbors (2D vectors in the buyer's reference frame) through the CommunicationSession's `get_distance_from` method, to be able to compute its distance to the location (for an example, see Fig. 8.2).

8.5.3 Step

The Behavior's `step` method is called by the Agent class, receiving sensors and an AgentAPI as arguments. Its role is to decide the robot's next movement, based on the sensor readings, Behavior's state and NavigationTable information. The decision for the robot's

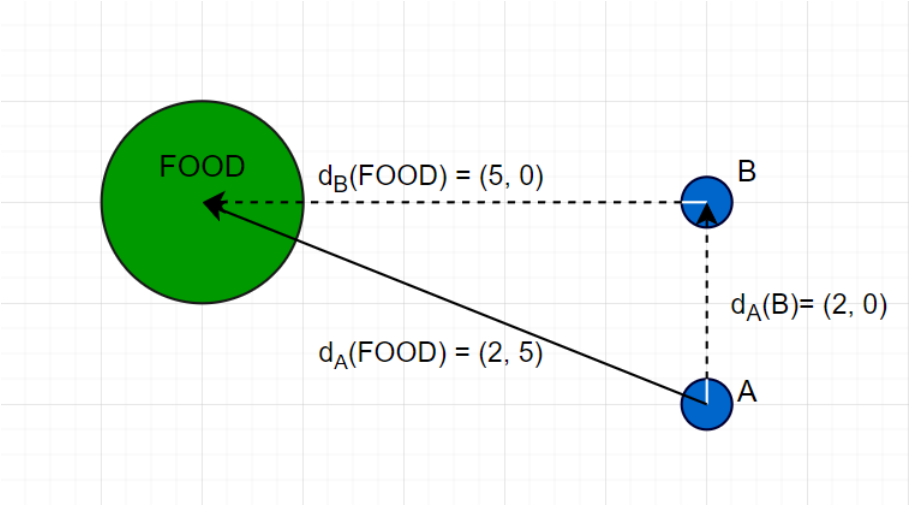

Figure 8.2: Example of A buying information from B about the FOOD location. B's distance to the food $d_B(FOOD) = (5, 0)$ is rotated to be in A's reference to $(0, 5)$, to which is added the distance to B in A's reference frame $d_A(B) = (2, 0)$, resulting in the final vector $d_A(FOOD) = (2, 5)$

movement is stored in the dr vector. Note $||dr||$ should be smaller or equal to the robot's maximum velocity V (this information can be obtained through the AgentAPI's `speed` method). If $||dr|| > V$, the Agent class uses V as the desired movement's length.

Furthermore, the targets inside the NavigationTable are rotated by the (negative) angle between the dr vector and the robot's local x axis, as the robot turns in the direction of motion. The robot assumes its instruction is carried out perfectly (i.e. as if an odometric reading confirmed the movement was carried out perfectly). In reality, the Agent class adds noise to the movement, resulting in the robot drifting away from its desired trajectory.

8.5.4 Get dr

The Behavior class' `get_dr` method should return the robot's desired movement in its own reference frame (see Fig. 8.3). In practice, the method simply returns the dr attribute updated by the `step` method.

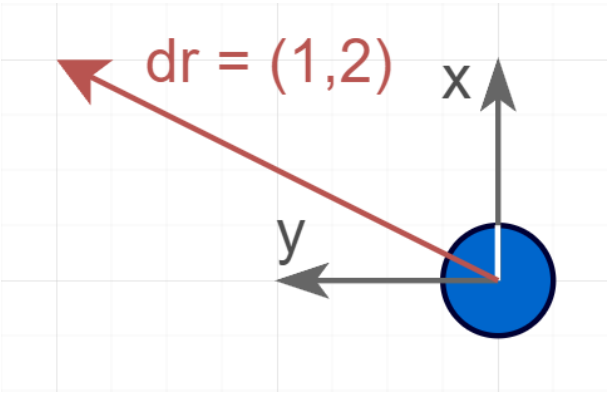

Figure 8.3: Example of the dr Vector in a Robot's Reference Frame

8.5.5 Get Target

The Behavior class' `get_target` method receives a location name as argument (i.e. FOOD or NEST) and returns the target the robot sells for that location. This method is called when another robot wishes to buy information from the given robot. The base HonestBehavior class returns the entry for that location in its NavigationTable. The SaboteurBehavior (and SmartboteurBehavior) rotates the distance attribute vector 90° from that entry, whereas the GreedyBehavior (and SmartGreedyBehavior) sets its age attribute to 1.

8.6 Payment

Transactions between robots buying and selling information are recorded in the PaymentDatabase. This is a centralized object, initialized by the Environment, abstracting away the idea of a decentralized database (i.e. the blockchain) that would be implemented in a physical robot swarm, but which is outside the scope of this work.

The PaymentDatabase keeps track of each robot's reward, as well as the transactions the robot is a buyer in. The transactions for each buyer are stored in the buyer's *PaymentSystem*. The PaymentSystem object calculates the amounts to be transferred from the buyer to seller according to the payment schemes described in Section 7.3.1. The only exception is the Fixed Price Transactions for which the transactions do not have to be recorded, since the price can instantly be deducted from the buyer's reward and added to the seller's reward.

For the other payment schemes relying on the sellers getting a share of a buyer's reward, the PaymentSystem class provides a `get_shares_mapping` method to compute what share of the total reward (initially awarded to the buyer when it deposits a strawberry in the nest) each seller receives.

8.7 Market

The Market class implements the reward mechanisms described in Section 7.3.2. It contains a `sell_strawberry` method that returns the reward R obtained for depositing a strawberry. The Market's `step` function is used to keep track of time, which is needed for the Round-trip Duration Reward mechanism.

8.8 Helper Modules

To finish this chapter, we quickly describe some helper modules whose functions are called in various parts of the rest of the code.

8.8.1 Random Walk

The `random_walk` module implements functions to calculate the probabilistic distributions used for the robots' random walk in their exploration state. In a random walk, two main characteristics of the movement are subject to randomness: the turning angle, and the amount of time or distance traveled between consecutive turns (also called step-length). The specific implementation of the random walk comes from an article [11] that provides two convenient parameters to control the step-length and turning angle distributions.

For the experiments conducted in this thesis, the search strategy is a hybrid between a correlated random walk (CRW) and a Lévy walk (LW). A correlated random walk means that there exists a correlation between a robot's consecutive turns, such that a robot is more likely to continue moving in the same general direction (i.e. it is biased toward low amplitude turning angles). A probability density function having such characteristics is a wrapped Cauchy distribution, which reads as:

$$f_\rho(\theta) = \frac{1}{2\pi} \frac{1-\rho^2}{1+\rho^2-2\rho\cos\theta}\,. \tag{8.1}$$

Furthermore, a pure CRW's step-length, or time between consecutive turns, follows a Gaussian distribution.

A Lévy walk is characterised by its heavy-tailed step-length distribution, following a power law,

$$P_\alpha(\delta) \sim \delta^{-(\alpha+1)}\,, \tag{8.2}$$

which in practice leads to series of quick turns (allowing local exploration of an area) followed by long straight-line displacements. A pure LW has a uniform turning angle distribution.

The random walk implemented for the simulations has a distribution of turning angles according to the CRW (with parameter $\rho = 0.9$), but a Lévy step-length distribution (with parameter $\alpha = 1.4$). The CRW distribution is represented by a finite set of 360 weights (one for each integer turning angle between 0° and 359°). When choosing a new turning angle, a robot samples one turning angle based on these weights (which can be obtained through the module's `get_crw_weights` function). In the same way, when choosing the next step-length, a robot samples it proportionally to the distribution of Eq. 8.2, where δ varies between 1 and 1000. This means the distribution is truncated and the probability to do more than 1000 steps is not taken into account.

8.8.2 Utils

The `utils` module provides various functions that do not belong in one of the other classes. It contains functions to rotate vectors or calculate their norm, as well as some Exceptions used in different parts of the rest of the code.

Part III

Analysis and Results

Chapter 9

Experimental Protocol

In this chapter, we quickly describe how the data used in the following chapters is gathered, and what parameters we use.

An experiment consists of running simulations for a fixed amount of τ_{max} time-steps each, with a population of N_r robots. At the end of every simulation, we record the number of items collected by each robot, as well as robots' individual wealth, in separate CSV files. In practice, to gather enough data, simulations are run 128 independent times (using a different random seed), with the same parameters. In the end, the "items collected" CSV file (the same can be applied to the "wealth" CSV file) contains 128 lines, each containing N_r values (one number of items collected or wealth for each robot). If the population consists of two different types of robots, with N_1 of the first type (the collaborative type by convention) and N_2 of the second type (the byzantine type by convention), the N_1 first values of each line refer to the first type of robots, whereas the N_2 last values of each line refer to the second type. The values for the parameters that are fixed for all experiments can be found in Tab. 9.1. Parameters specific to a given experiment will be explicitly mentioned when showing the results.

Parameter Description	**Symbol**	**Value**
Number of robots	N_r	25
Simulation length (total time-steps)	τ_{max}	15 000
Environment width×height	$W \times H$	1200×600
Food patch position	P_F	(200, 300)
Nest position	P_N	(1000, 300)
Food and Nest sites radii	r_F, r_N	50
Robot communication range	r_C	50
Robot maximum velocity	V	2.5
Robot drift bias distribution parameters (Eq. 5.1)	m_μ and s_μ	0.05 and 0.05
Robot drift standard deviation	σ	0.05
Robot initial reward	R_0	3
Correlated random walk parameter	ρ	0.9
Lévy walk parameter	α	1.4

Table 9.1: Base Simulation Parameters

Chapter 10

Individual Protection

In this chapter, we first show how the performance of a system composed of *naive* robots (the base behavior described in Section 6), heavily deteriorates when we introduce *saboteurs* (see Section 7.1) in the population. We then show how by introducing more skeptical robots (i.e. the *careful* and *smart* behaviors described in Section 7.2), the population is more robust to saboteurs. However, we also show that this robustness has a cost, as more skeptical robots are also less efficient in a population without saboteurs. For the moment, we consider robots exchange information for free (i.e. there is no payment system).

10.1 Naive Behavior

10.1.1 Honest Population

As a base case, we can analyze the performance of a population of naive robots implementing the social navigation strategy described in Section 6. To do so, we run 128 simulations with 25 naive robots. The results are shown in Fig. 10.1.

From the violin plot on the left, we can see the median robot collects 21 items per run. We can also see 50% of robots (first to third quartile) collect between 19 and 21 items per run. The distribution resembles a normal distribution, negatively skewed (tail towards low values). This is logical since inaccurate robots (with a large drift) can stray from the main robot chain for a long time, collecting few items, but very accurate robots (with very low drift), are not much more efficient than the rest of the robot chain. In other words, all robots with moderate levels of drift have a similar performance because, thanks to frequent updates and corrections of their information from other robots, they form a chain and are able to efficiently navigate between the food and nest area. Therefore there is no considerable difference in number of collected items among the the upper half of the group. Instead, there is a large drop in performance when robots have drifts so large that they fail to remain in the robot chain and drift away.

10.1.2 Population with Saboteurs

For a second set of experiments, we analyze the performance of a population of 24 naive robots with 1 saboteur robot. The saboteur behaves like the naive, except it falsifies information it gives to other robots by rotating its 2D distance vector attribute by 90° (see Section 7.1).

The plots from Fig. 10.2 first show that the performance of the naive robots heavily drops when a saboteur is present, and is also more spread out. From the violin plot we

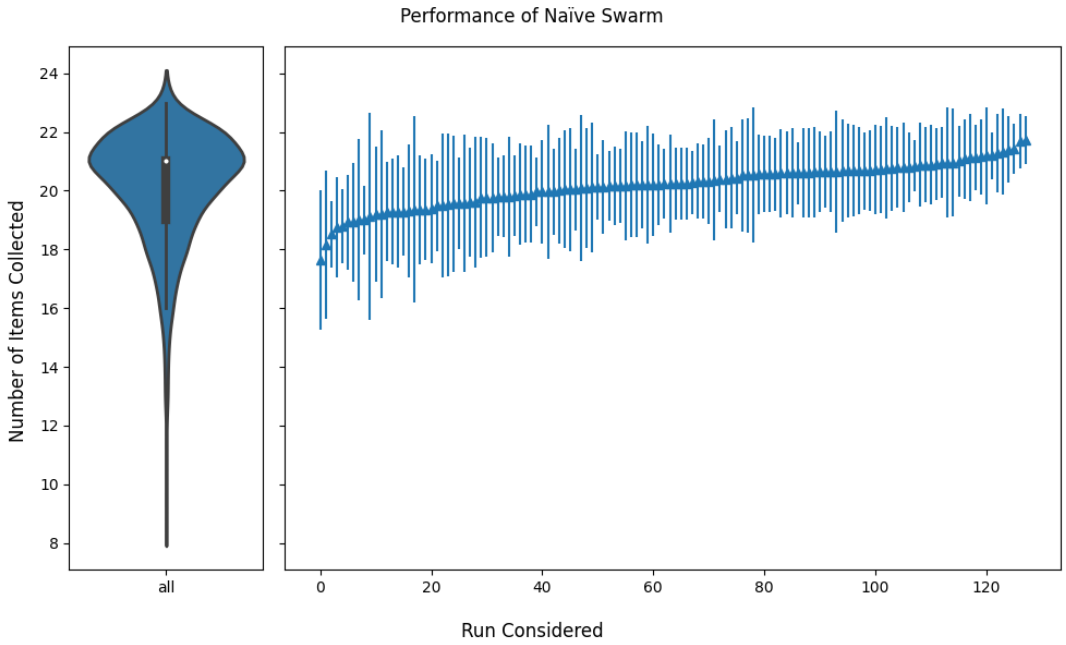

Figure 10.1: Performance of a swarm of $N_1 = 25$ naive robots, by measuring the number of items collected by each robot at the end of a run. The violin plot on the left aggregates the data from all runs. It plots a smoothed distribution of the number of items collected by each of the 3200 robots (25 robots $\times$ 128 simulations). Inside the violin plot, we can see a miniature box plot, with the white point being the median number of items collected across all 3200 robots. The figure on the right plots the mean and standard deviation of the individual robot rewards, for each of the 128 runs, sorted by increasing mean. This allows us to analyze the data by individual run, to visually notice effects possibly hidden by the aggregation.

can see the first to third interquartile range is between 6 and 11 items collected (median of 9), compared to 19 to 21 items collected (median of 21) for a fully cooperative population (composed of naive robots only). We can also see the performance of the saboteur robots is on average better than the naives' (interquartile range between 11 and 15, median of 13).

From the plot on the right of Fig 10.2, we can also see the means of the naive sub-population spread from 4 to 15 items collected across the 128 runs, which varies much more than for a fully cooperative population of naive robots (between 18 and 22 in the right plot of Fig 10.1). Finally, this plot allows us to see a clear negative correlation between the performance of the lone saboteur and the other naive robots in each run. When the saboteur performs well, it means that it is able to remain on the path between food and nest for a longer period of time, and in that case, the naive group collects very few items and often drifts away from the main path. Instead, when the saboteur performs poorly and often drifts away from the main path, the naive group tends to perform better as the robot chain is less often disrupted.

10.2 Increased Skepticism

10.2.1 Performance Against Saboteurs

As Section 10.1.2 shows, the number of items collected by a naive population heavily deteriorates when a saboteur is present. In this section we show how the careful and smart behaviors described in Section 7.2 can make the swarm more robust to saboteurs.

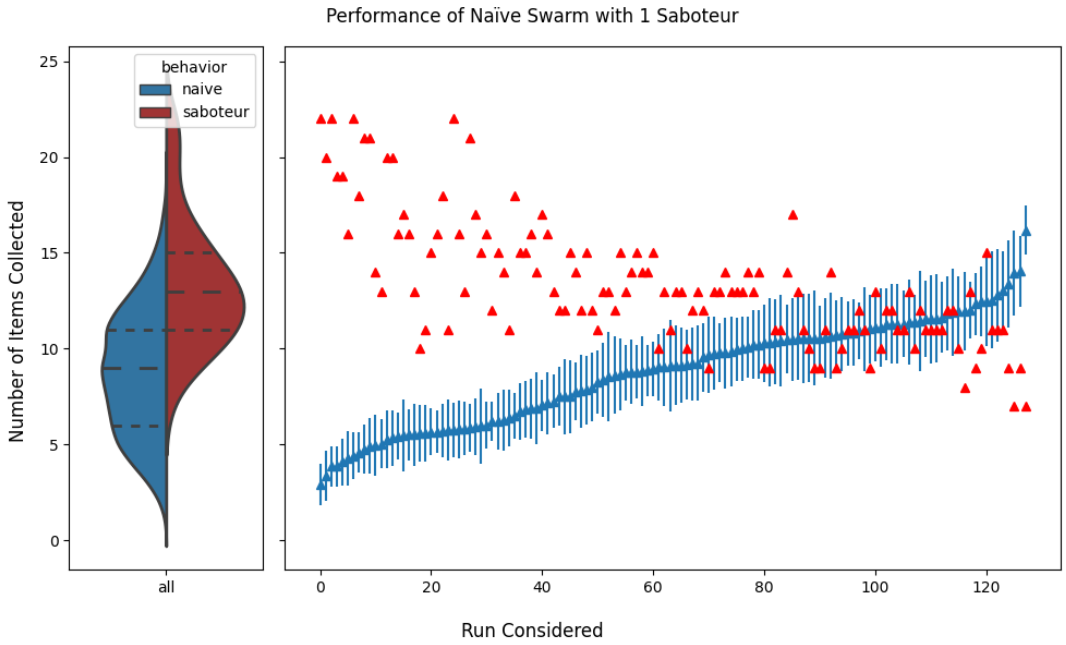

Figure 10.2: Performance of a swarm of $N_1 = 24$ naive robots and $N_2 = 1$ saboteur, by measuring the number of items collected by each robot at the end of a run. The violin plot on the left shows the distributions of items collected for the aggregated 3072 naive robots (in blue) and 128 saboteurs (in red). The dashed lines in the violin plot show the first and third quartiles (many small dashes), as well as the median (few long dashes). The plot on the right shows the mean and standard deviation for the items collected by each robot for each run separating between the naive subgroup (in blue) and saboteur individual (in red), sorted by increasing means of the naive subgroup. We can clearly see a negative correlation between the saboteur's performance and the rest of the naive robots in each run. When the saboteur performs well, the naive robots perform badly, and vice versa.

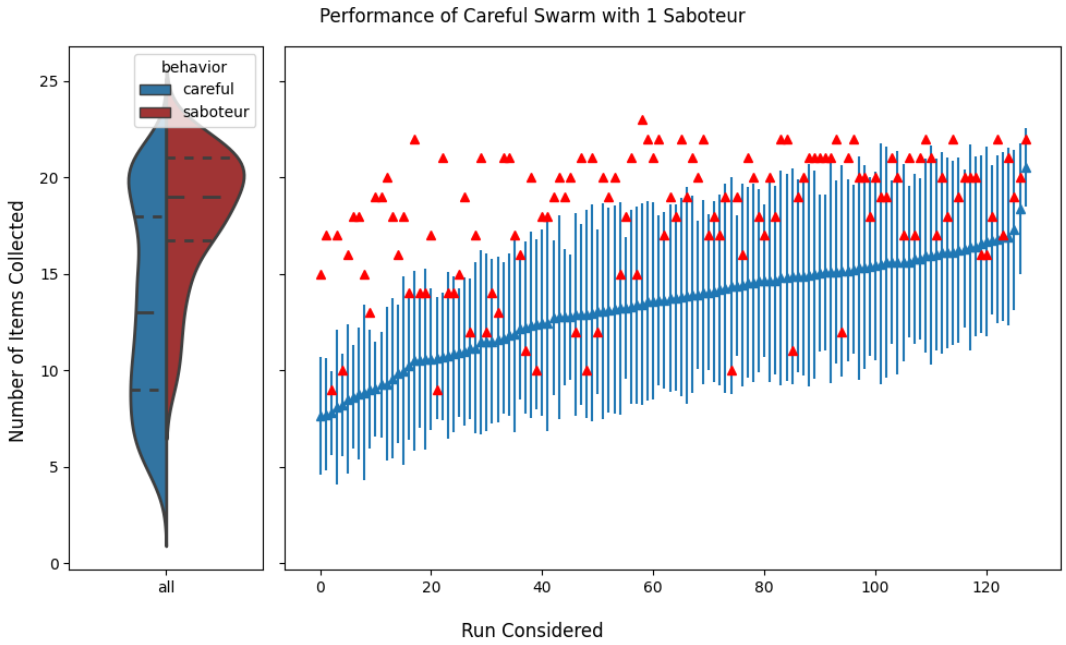

Figure 10.3: Performance of a swarm of $N_1 = 24$ careful robots with security level $s = 3$ robots and $N_2 = 1$ saboteur.

Both of these behaviors are more skeptical than naive robots, in the sense that they wait until having acquired multiple pieces of information before changing their beliefs.

Fig. 10.3 and Fig. 10.4 respectively show the performance of swarms of 24 careful robots and 24 smart robots, in the presence of a single saboteur. We can see from the violin plots that both strategies are able to collect more items in the presence of a saboteur, than their naive counterpart can (Fig. 10.2). For both careful and smart behaviors, we can also see that the clear negative correlation between the honest subgroup's performance and the saboteur individual has disappeared.

From the violin plots, we can also see that the median number of items collected by a careful robot in the presence of a saboteur is about 45% higher than for a naive (13 compared to 9), and for smart robots, the gain in efficiency is higher than 75% (median of 16 items compared to 9).

However, for both skeptical behaviors, the interquartile range is also wider than in the case of naive robots ([9,16] for the careful, [12, 20] for the smart, compared to [6,11] for the naive). Furthermore, we can note that the performance of the saboteur individuals increases thanks to the robustness of the honest subgroup. Indeed, saboteurs are also more efficient when the dynamic chain formed by the honest robots holds, as saboteurs rely on honest information for their own navigation. Finally, we can see that saboteurs perform as well when surrounded by careful robots as smart robots (same first quartile, median, and third quartile in both violin plots).

All in all, we can conclude that the smart behavior is the most robust of the three honest behaviors, when facing a saboteur in the swarm.

10.2.2 The Cost of Robustness

In the previous section (Section 10.2.1), we show how the previously introduced careful and smart behavior especially, are able to increase the swarm's robustness to the saboteur behavior (i.e. maintain its performance compared to the naive behavior). In this section, we show how this robustness comes at the cost of a lower efficiency when the swarm only

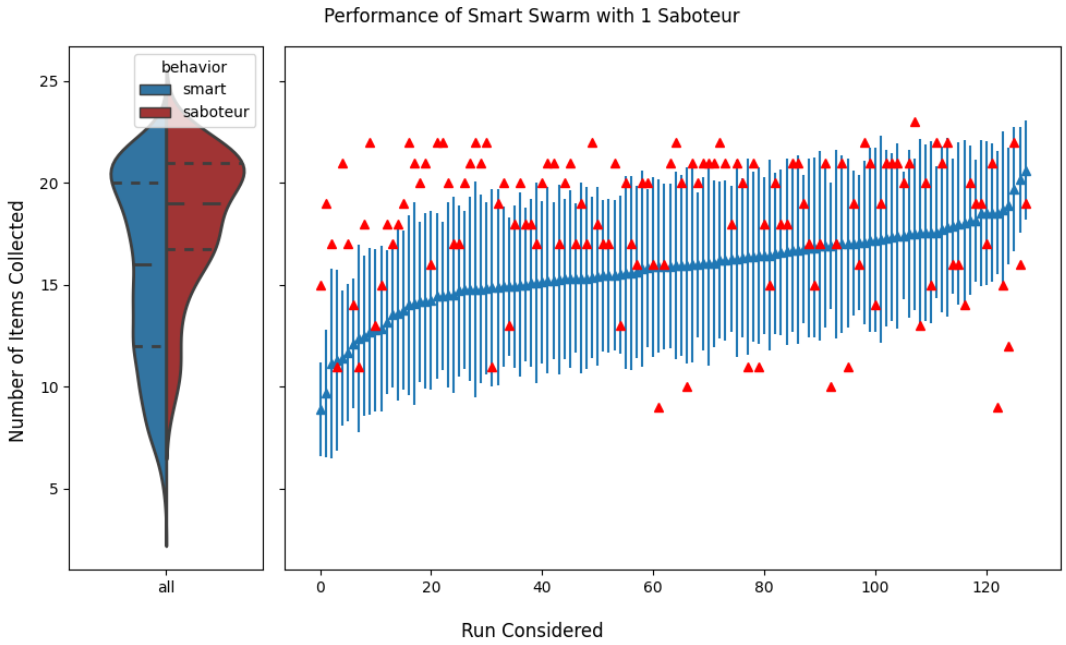

Figure 10.4: Performance of a swarm of $N_1 = 24$ smart robots with threshold $\theta = 0.25$ and $N_2 = 1$ saboteur.

contains honest robots (i.e. naive, careful, or smart).

Fig. 10.5 shows the violin plots for 3 experiments of 128 runs with 25 robots of each honest type. We can see naive robots perform the best, with a distribution concentrated around its median of 21 items collected per robot per run. Second come the smart robots with a wider distribution, and a median of 17 items collected. The careful robots perform the worst with the widest distribution and a median of 13 items collected per run. This means that the two skeptical behaviors are less efficient than the naive behavior when the population is fully cooperative. Security comes with a cost.

Furthermore, the two skeptical behaviors' median performance and their overall distributions are extremely similar in a fully cooperative setting, to when a saboteur is present in the population. This indicates that the two behaviors are able to filter out the saboteur's information effectively, unlike the naive behavior, who suffers a large drop in performance in the presence of a saboteur.

Finally, we can also see that the smart robots are more efficient than the careful robots in a fully cooperative environment. Since the smart behavior is superior to the careful behavior both in the presence and absence of saboteurs, we can conclude smart robots are strictly superior to careful robots. In the next section, we will thus only be considering the smart behavior.

10.3 Summary of the Individual Protection Analysis

By analysing the naive, careful, and smart behaviors' performance both in the context of a fully honest population and in a population with a single saboteur, we obtain the following main findings:

1. The naive robots' performance collapses in the presence of a single saboteur. In that scenario, saboteurs successfully disturb the system, and even perform better than the subgroup of naive robots.

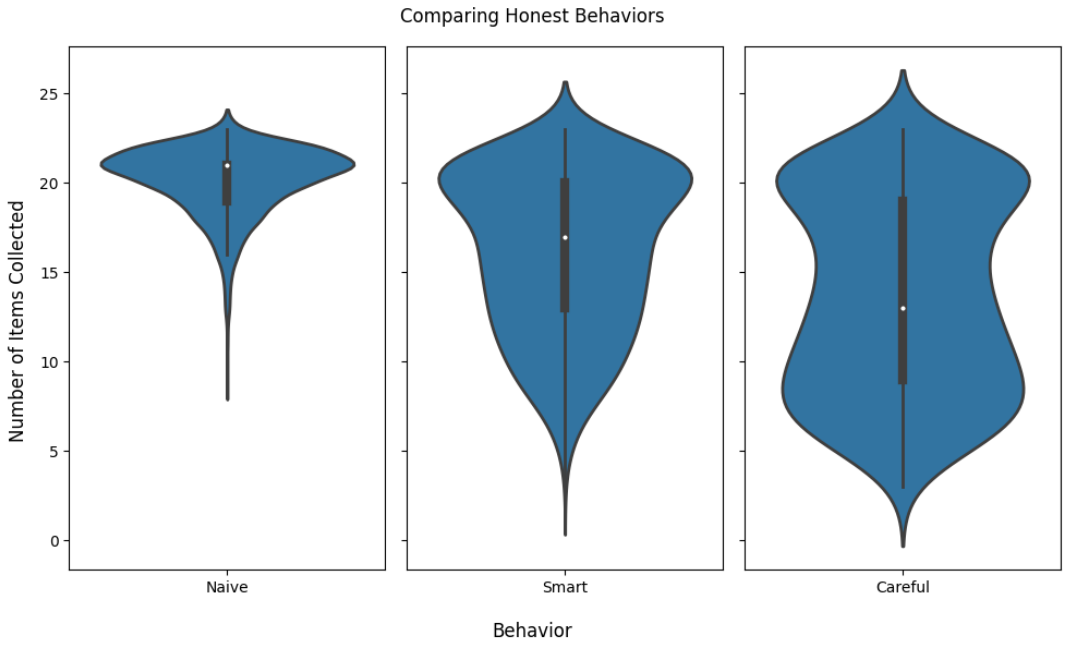

Figure 10.5: Performance of a swarm of $N_1 = 25$ robots of the three types of honest behaviors: naive, careful and smart. For the smart behavior, $\theta = 0.25$. For the careful behavior, $s = 3$.

2. The careful and smart behaviors are indifferent to a saboteur being present in the population, which means the system is more robust. However, this robustness comes at the cost of a reduced efficiency when no saboteur is present.

3. The careful behavior is strictly worse than the smart behavior.

Chapter 11

Systemic Protection

In the previous chapter, we show how the smart robots described in Section 7.2 are resilient to a saboteur's misinformation. Nonetheless, smart robots are less efficient than honest robots in a fully cooperative context, and we also show that, even though smart robots are resilient to saboteurs, a saboteur collects more items on average than a smart robot.

In this chapter we compare different payment systems and reward mechanisms (described in Sections 7.3.1 and 7.3.2 resepectively) as systemic rules to differentiate smart robots from Byzantines. Indeed, the goal is to find a combination of information payment system and reward mechanism, that causes saboteurs to be less wealthy than smart robots. Under this hypothesis, we could then explore in future work how using this wealth value can measure the trustworthiness of a robot, to implement new decision rules when choosing whether to buy information from another robot, and possibly improve the system's performance (increase the number of items collected in the presence of Byzantines).

To be clear, we are not interested in measuring a robot's performance (i.e. the number of items that it collected). Instead, we look at each robot's wealth, that it accumulates through depositing items at the nest site (i.e. the reward mechanism) and by selling information to other robots (i.e. the payment system). Furthermore, since the reward's value can be chosen arbitrarily, the numeric value of a robot's total wealth does not carry much meaning on its own. Instead, by computing the fraction of a single robot's wealth compared to the whole population (i.e. the sum of the wealth of every robot in the population), we can compare different combinations of payment systems and reward mechanisms. While the total amount of money injected into the system by the reward mechanism may not be the same, our analysis allows us to study the efficacy of system protection against dishonest behaviors.

Finally, in this section we use a saboteur behavior exhibiting skepticism (i.e. saboteurs perform the same information comparison procedure as smart robots before updating their target information). This allows us to explore what happens when more than one saboteur is present in the population. Otherwise, if saboteurs simply buy information like naive robots, smart robots can often ignore saboteurs' information, whereas saboteurs sabotage each other.

11.1 Reward Share Transactions

In this section we explore how the Reward Share Transactions payment system described in Section 7.3.1 shapes the wealth distribution. The hypothesis is that the seller has an incentive to sell accurate information in order to let the buyer complete a round-trip quickly, because the seller only receives a share of the buyer's reward once the buyer

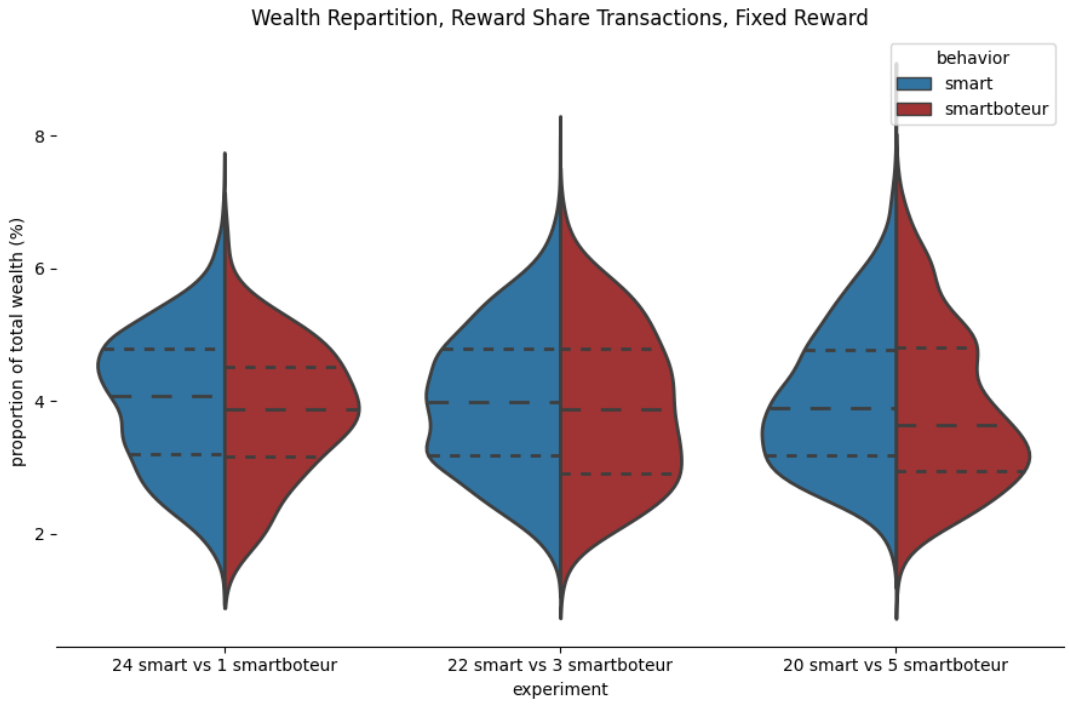

Figure 11.1: Wealth repartition using the Reward Share Transactions system with Round-trip Duration Reward mechanism, for swarms of $N_1 = [24, 22, 20]$ smart robots and $N_2 = [1, 3, 5]$ smart saboteurs. Similarity threshold $\theta = 0.25$. Reward share $a = 50\%$. Violin plots obtained from 128 runs for each of the three N_1 (and corresponding N_2) values.

deposits its strawberry in the nest. Being more efficient and depositing more items also reward the seller who gets paid more frequently. Furthermore, if a saboteur sends a robot in the wrong direction, that robot will probably have to buy more information to find its way, reducing the saboteur's share (since more sellers share a fixed fraction of the buyer's reward).

11.1.1 Paired with Fixed Reward Mechanism

First we pair the Reward Share Transactions payment system with the Fixed Reward mechanism (see Section 7.3.2). The results can be found in Fig. 11.1. We compare the distribution of the fraction of the individual robot's wealth over the total population wealth at the end of a simulation. To be precise, for each of the 128 runs, the value of each robot's final wealth is recorded at the end of the simulation. We then compute each robot's fraction of the total wealth in its population for that run (for example, if every of the $N = 25$ robots has the same wealth at the end of a simulation, they all have a fraction $\frac{1}{25} = 4\%$ of the total wealth). We then plot the distribution of these fractions using a violin plot, separating between smart honest robots, and smart saboteur robots (we will simply refer to honest robots and saboteurs from now on). The objective of this analysis is to test if the payment system and reward mechanism lead to honest robots being richer than saboteurs.

We can see in Fig 11.1 that the reward fraction is very similar between honest and saboteur robots. This means that our hypothesis is falsified and the Reward Share Transactions payment system paired with the Fixed Reward mechanism does not distribute more wealth to honest robots. This can be explained by the fact that the smart robots oftentimes buy a piece of saboteur information and do not use it actively afterward due

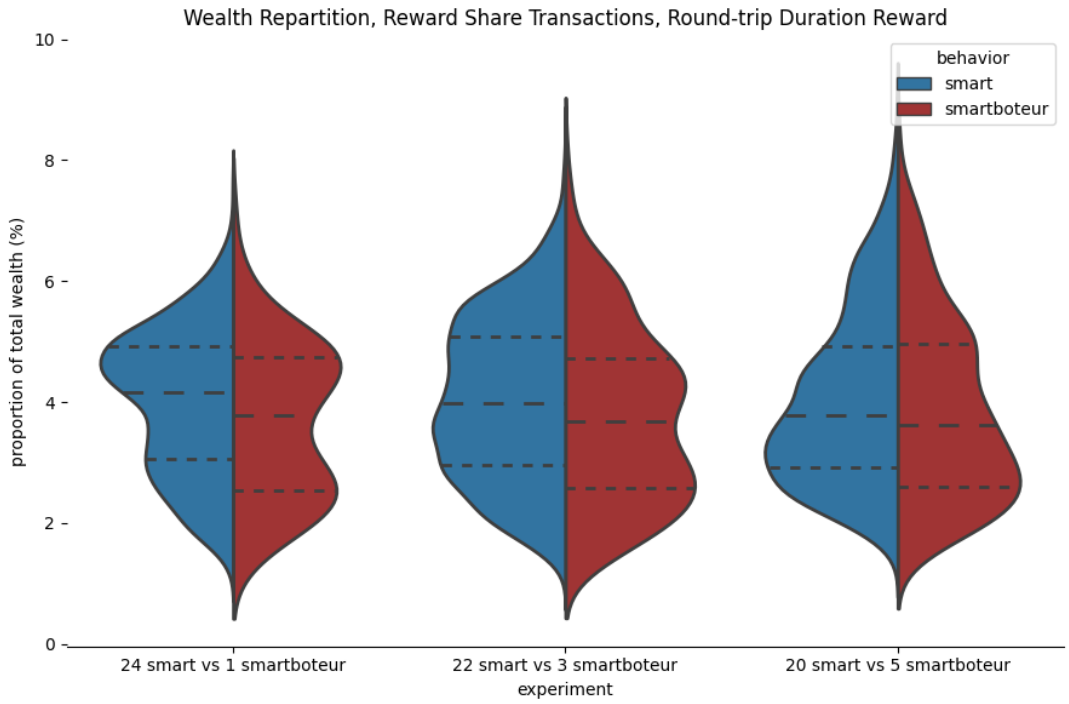

Figure 11.2: Wealth repartition using the Reward Share Transactions system with Round-trip Duration Reward mechanism, for swarms of $N_1 = [24, 22, 20]$ smart robots and $N_2 = [1, 3, 5]$ smart saboteurs. Similarity threshold $\theta = 0.25$. Reward share $a = 50\%$. Violin plots obtained from 128 runs for each of the 3 N_1 (and corresponding N_2) values.

to their "smart" way of filtering information, but the payment system still awards the saboteur information the same share as truthful, useful, information sold by honest robots.

11.1.2 Paired with Round-trip Duration Reward Mechanism

To try and punish the saboteurs more heavily, we use a reward mechanism that distributes a higher reward to robots that spend less time collecting and depositing a strawberry. The rationale is that in this way, when a saboteur succeeds in sending a robot in the wrong direction, it will increase that robot's round-trip duration, diminishing that robot's end reward, leading to the saboteur getting a small share.

Unfortunately, from Fig. 11.2, we can see this hypothesis does not materialize. We can provide the following considerations as an explanation:

1. When a saboteur's information is ignored it is still paid as much as other honest robots whose information is used by the buyer, just like in the Fixed Reward case.

2. When a saboteur's information is used, the robot is sent off in the wrong direction. This leads to a longer round-trip duration, and a smaller reward for both buyer and collective sellers. This means that all the sellers that sold information for the same round-trip as the saboteur are also penalized. Therefore, the saboteur is not punished more than the honest robots by the lower reward share it gets from the payment system due to increased round-trip time.

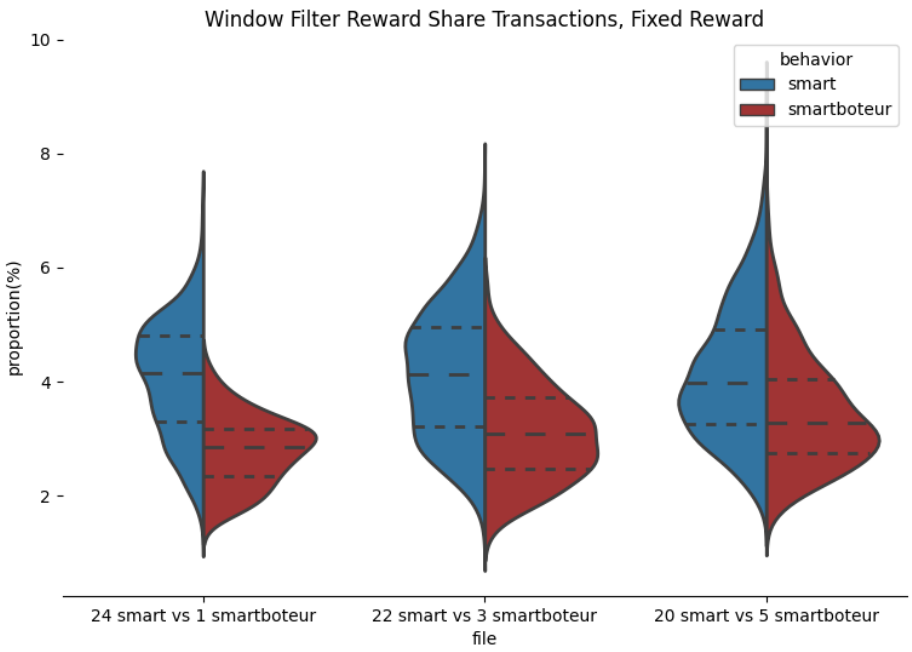

Figure 11.3: Wealth repartition for swarms of $N_1 = [24, 22, 20]$ smart robots and $N_2 = [1, 3, 5]$ smart saboteurs. Similarity threshold $\theta = 0.25$. Reward share $a = 50\%$. Violin plots obtained from 128 runs for each of the 3 N_1 (and corresponding N_2) values.

11.2 Window Filter Reward Share Transactions

The previous section shows that saboteurs are hard to punish, since the decision not to use their information happens after the transaction has concluded. This means they are paid regardless of the use or the discarding of their information by the buyer. What we want is a system that can detect saboteur information, and reward it less than honest information.

This is the idea behind the Window Filter Reward Share Transactions described in Section 7.3.1, and inspired by previous work on blockchain-secured robot swarms that employed an outlier detection smart contract to reduce the reward of false information when robots monitored an environment [31]. Here, when a robot buys a target, the buyer can agree with the seller on how much the buyer would need to rotate to go where the target points to. Due to a robot's drift, we can expect honest information to readjust a robot's orientation by a similar amount at each transaction. Since saboteur information would make the robot turn a completely different angle, and since we assume saboteurs to be a minority in the population, their transactions would figure as outliers. By rewarding similar information more than outliers, we expect the saboteurs to receive smaller shares for their information sales.

The results for the Window Filter Reward Share Transactions system paired with both Fixed Reward and Round-trip Duration Reward mechanisms can be found in Fig. 11.3 and 11.4 respectively. With both reward mechanisms, we can see a clear difference between the proportion of total wealth attributed to honest robots and saboteurs. This means this payment system is able to punish saboteurs. We can also see the difference between the honest group's distribution and the saboteur's group distribution decreases as the proportion of saboteurs in the population increases. This is logical since, as the proportion of saboteurs increase, the proportion of similar outlier transactions during a round-trip

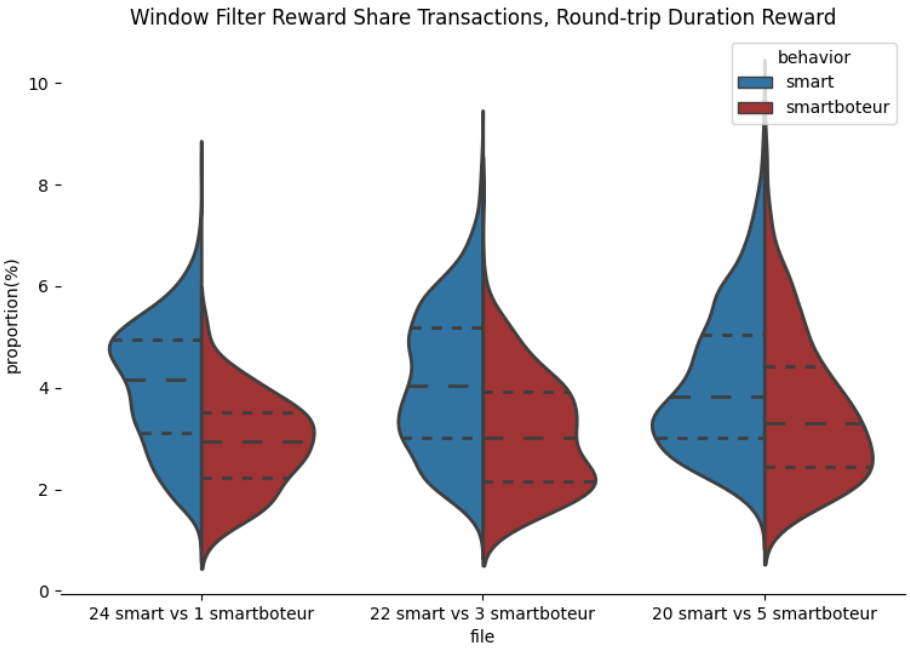

Figure 11.4: Wealth repartition for swarms of $N_1 = [24, 22, 20]$ smart robots and $N_2 = [1, 3, 5]$ smart saboteurs. Similarity threshold $\theta = 0.25$. Reward share $a = 50\%$. Violin plots obtained from 128 runs for each of the 3 N_1 (and corresponding N_2) values.

also increases, leading to all outliers obtaining a bigger fraction of the reward when the buyer deposits a strawberry.

11.3 Greedy Behavior

In the previous section we show how using the Window Filter Reward Share Transactions payment system can reward honest robots more than saboteurs. In this section, we show how another type of Byzantine behavior, the greedy robot (see Section 7.4), is able to exploit this payment system, to obtain a high fraction of the total wealth.

Fig. 11.5 shows that greedy robots are able to accrue a much bigger proportion of the total wealth at the end of a simulation, than the honest robots can. By lying on the target's age attribute, greedy robots are involved as sellers in a lot of transactions, and thus get a significant portion of each buyer's reward when it deposits an item.

Trying to counter the greedy behavior is not tackled in this thesis, but we propose the following possible approaches:

1. Since greedy behaviors rely on selling a lot of information, one could modify the Window Filter Reward Share Transactions payment system as to not reward, or even punish, sellers than sell more than a certain number of targets to the same robot.

2. Since in practice (i.e. for physical robots), the payment database would be implemented using a blockchain and smart contracts, one could also use these tools to make robots agree on a shared global clock or time-keeping service, to record the time at which information is collected and sold. The age of an information target could then be computed as the difference between two cryptographically verified

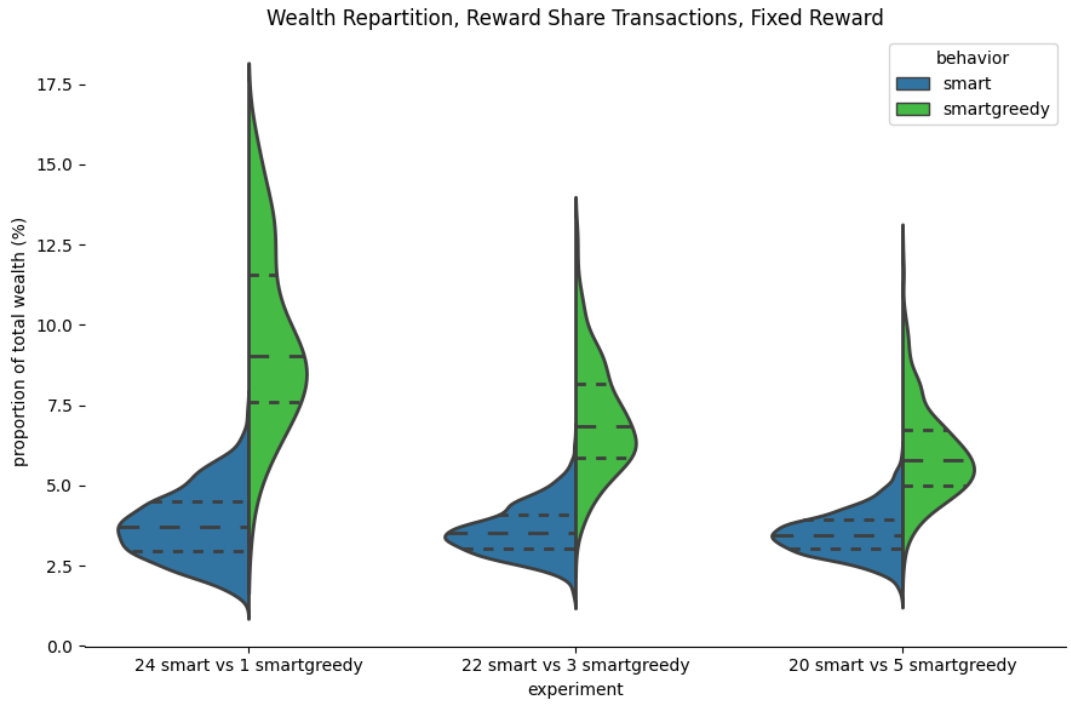

Figure 11.5: Wealth repartition for swarms of $N_1 = [24, 22, 20]$ smart robots and $N_2 = [1, 3, 5]$ smart greedy robots. Similarity threshold $\theta = 0.25$. Reward share $a = 50\%$. Violin plots obtained from 128 runs for each of the 3 N_1 (and corresponding N_2) values.

timestamps. Since in this framework, greedy robots could not lie on a target's age, they would not be able to exploit the payment system.

11.4 Summary of the Systemic Protection Analysis

In this chapter, we show how several payment systems and reward mechanisms combinations distribute wealth in a population of honest and saboteur robots. We show that the Window Filter Reward Share Transactions payment system provides a mechanism to distinguish between honest and saboteur robots. Under this system, a robot's reward would be an indication of how trustworthy that robot is. By creating a new decision rule (i.e. to decide whether to buy information) based on the seller's reward, instead of solely on the target information's age, future work could investigate how to increase the swarm's performance (in number of items collected) in the presence of Byzantines, more than the individual protection measures we present in Section 7.2.

We also show how the payment system itself, despite being effective against saboteurs, can be exploited by another type of Byzantine behavior (i.e. the greedy behavior). We do not provide any concrete solution to this problem but present ideas to be tested in future work.

Chapter 12

Future Work and Conclusion

12.1 Extending this Work

12.1.1 Window Filter Reward Share Transactions

In Section 11.2, we show that the Window Filter Reward Share Transactions system is an effective way to distribute more wealth to honest robots than to saboteurs. However, we have yet to use this discrepancy in a practical manner, to be able to improve the swarm's efficiency, as measured by the number of items collected at the end of a simulation (as it is the goal in the end). To do so, we suggest solutions that can be grouped into two main categories:

1. Use a seller's wealth as a decision variable when choosing whether to buy information from that robot. Indeed, since honest robots accumulate more wealth on average than Byzantines, buying information from wealthy robots increases the probability the information is truthful. Furthermore, this could create a positive feedback loop, where wealthy honest robots get richer and saboteurs poorer, gradually increasing the divide between the two, but this would need to be verified experimentally.

2. Use a robot's wealth, or the Window Filter directly, to identify and blacklist saboteurs, analogously to previous work in collective decision making [31]. The blacklist would be computed and stored through the blockchain and allow robots to specifically avoid buying information from saboteurs.

Finally, one of the drawbacks of the Window Filter Reward Share Transactions system is that it needs to compute the number of similar transactions for each transaction recorded in the last round-trip. This can cost a number $O(|T|^2)$ (where $|T|$ is the number of recorded transactions) of operations, and be quite slow and computationally expensive. Since the idea is to attribute higher shares of a buyer's reward to similar information, we can imagine using a different similarity measure than the count of transactions similar to the one considered (such as the inverse distance to the median orientation of all transactions for example). Again, this would have to be tested experimentally.

12.1.2 Dealing with Greedy Robots

One of the main drawbacks of the Window Filter Reward Share Transactions is that it can be exploited by greedy robots or any behavior that finds a way to sell a lot of information to other robots. In Section 11.3 we propose two general ways to mitigate this exploit. For the first proposed solution, we can take inspiration from how Sybil attacks

can be mitigated using blockchain technology [31], by making sellers pay a fixed cost for any transaction recorded in the blockchain. By choosing this cost carefully (which may depend on the size of the swarm), we could prevent greedy robots from selling too much information, as these fixed costs would accumulate and end up being a larger amount than the total share of the reward distributed by the Window Filter Reward Share Transactions to all sellers.

12.1.3 Exploring Other Ideas

Even though in this thesis we concentrate on the impact (and its mitigation) of Byzantines in a swarm of honest robots, the simulator has been built to easily explore other properties of the information market we propose. For example one could:

- investigate what happens when communication between robots is costly (either monetarily, or if robots have to stop moving to communicate);
- develop new robot behaviors that exploit in new ways the Window Filter Reward Share Transactions system;
- look into heterogeneous swarms of two different types of honest robots with different capabilities (for example, one type may not be able to pick up items), and how wealth could indicate the individual contribution of each type of robot to the common foraging task (e.g. by allocating part of the robots to become static beacons that only sell accurate target information to other robots transporting the objects).

For the first idea, everything is in place to test different scenarios without writing additional code. For the last two ideas, one would simply need to extend the simulator with new Behavior classes, which can be easily implemented following the design of existing behaviors.

12.2 Conclusion

In this thesis, our goal was to create a simulator to explore the ramifications of using an information market to regulate communication between robots performing a social navigation task. We especially focused our analysis on the performance of honest robots in the presence of Byzantines, and proposed two complementary classes of protection measures against the latter. Through increased skepticism, we show how a swarm of smart robots is more robust to the disruption caused by Byzantines, but also how this robustness comes at the cost of reduced efficiency when all robots are cooperative. Through the Window Filter Reward Share Transactions payment system, we show how Byzantines accumulate less wealth than honest robots and we suggest concrete mechanisms to translate these results into ways to further improve the swarm's resiliency.

Bibliography

[1] A. Aswale, A. López, A. Ammartayakun, and C. Pinciroli. Hacking the colony: On the disruptive effect of misleading pheromone and how to defend against it. *CoRR*, abs/2202.01808, 2022.

[2] F. Bartumeus, M. G. E. da Luz, G. M. Viswanathan, and J. Catalan. Animal search strategies: A quantitative random-walk analysis. *Ecology*, 86(11):3078–3087, 2005.

[3] R. Beckers, J. Deneubourg, and S. Goss. Modulation of trail laying in the ant *Lasius niger* (Hymenoptera: Formicidae) and its role in the collective selection of a food source. *Journal of insect behavior*, 6(6):751–759, 1993.

[4] P. Blackwell. Random diffusion models for animal movement. *Ecological modelling*, 100(1):87–102, 1997.

[5] M. D. Breed and J. Moore. Chapter 9 - foraging. In M. D. Breed and J. Moore, editors, *Animal Behavior (Third Edition)*, pages 309–341. Academic Press, San Diego, 2022.

[6] V. Buterin. A next-generation smart contract and decentralized application platform. ethereum project white paper. Technical report, Ethereum Foundation. `https://ethereum.org/en/whitepaper/`, 2014. Accessed on 06/02/2022.

[7] A. Campo, A. Gutiérrez, S. Nouyan, C. Pinciroli, V. Longchamp, S. Garnier, and M. Dorigo. Artificial pheromone for path selection by a foraging swarm of robots. *Biological cybernetics*, 103(5):339–352, 2010.

[8] F. Canciani, M. S. Talamali, J. A. Marshall, T. Bose, and A. Reina. Keep calm and vote on: Swarm resiliency in collective decision making. In *Proceedings of workshop resilient robot teams of the 2019 IEEE International Conference on Robotics and Automation (ICRA 2019)*, page 4, 2019.

[9] E. A. Codling, M. J. Plank, and S. Benhamou. Random walk models in biology. *Journal of the Royal Society interface*, 5(25):813–834, 2008.

[10] G. De Masi, J. Prasetyo, E. Tuci, and E. Ferrante. Zealots attack and the revenge of the commons: Quality vs quantity in the best-of-n. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12421 of *Lecture Notes in Computer Science*, pages 256–268. Springer International Publishing, Cham, 2020.

[11] C. Dimidov, G. Oriolo, and V. Trianni. Random walks in swarm robotics: An experiment with Kilobots. In *Lecture Notes in Computer Science*, pages 185–196. Springer International Publishing, 2016.

[12] F. Ducatelle, G. A. Di Caro, C. Pinciroli, and L. M. Gambardella. Self-organized cooperation between robotic swarms. *Swarm intelligence*, 5(2):73–96, 2011.

[13] F. Ducatelle, G. A. Di Caro, C. Pinciroli, F. Mondada, and L. Gambardella. Communication assisted navigation in robotic swarms: Self-organization and cooperation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4981–4988, 2011.

[14] R. Fujisawa, S. Dobata, K. Sugawara, and F. Matsuno. Designing pheromone communication in swarm robotics: Group foraging behavior mediated by chemical substance. *Swarm intelligence*, 8(3):227–246, 2014.

[15] S. Goss, S. Aron, J. Deneubourg, and J. Pasteels. Self-organized shortcuts in the Argentine ant. *Die Naturwissenschaften*, 76(12):579–581, 1989.

[16] N. Hoff, R. Wood, and R. Nagpal. *Distributed Colony-Level Algorithm Switching for Robot Swarm Foraging*, pages 417–430. Springer Berlin Heidelberg, 2013.

[17] D. L. Johnson, N. Ntlatlapa, and C. Aichele. Simple pragmatic approach to mesh routing using BATMAN. In *Proceedings of the 2nd IFIP International Symposium on Wireless Communications and Information Technology in Developing Countries (WCTID 2008)*, 2008.

[18] T. Kazama, K. Sugawara, and T. Watanabe. Collecting behavior of interacting robots with virtual pheromone. In *Distributed Autonomous Robotic Systems 6*, pages 347–356. Springer Japan, Tokyo, 2007.

[19] A. A. Khaliq, M. Di Rocco, and A. Saffiotti. Stigmergic algorithms for multiple minimalistic robots on an RFID floor. *Swarm intelligence*, 8(3):199–225, 2014.

[20] R. Mayet, J. Roberz, T. Schmickl, and K. Crailsheim. Antbots: A feasible visual emulation of pheromone trails for swarm robots. In *Lecture Notes in Computer Science*, volume 6234, pages 84–94. Springer Berlin Heidelberg, 2010.

[21] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, 2008. (Accessed on 06/02/2022).

[22] O. Olsson, J. S. Brown, and K. L. Helf. A guide to central place effects in foraging. *Theoretical Population Biology*, 74(1):22–33, 2008.

[23] A. Pacheco, V. Strobel, and M. Dorigo. A blockchain-controlled physical robot swarm communicating via an ad-hoc network. In M. Dorigo, T. Stützle, M. J. Blesa, C. Blum, H. Hamann, M. K. Heinrich, and V. Strobel, editors, *Swarm Intelligence*, pages 3–15, Cham, 2020. Springer International Publishing.

[24] V. V. Palyulin, A. V. Chechkin, and R. Metzler. Lévy flights do not always optimize random blind search for sparse targets. *Proceedings of the National Academy of Sciences*, 111(8):2931–2936, 2014.

[25] C. S. Patlak. Random walk with persistence and external bias. *The Bulletin of Mathematical Biophysics*, 15(3):311–338, 1953.

[26] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo. Argos: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm intelligence*, 6(4):271–295, 2012.

[27] G. Primiero, E. Tuci, J. Tagliabue, and E. Ferrante. Swarm attack: A self-organized model to recover from malicious communication manipulation in a swarm of simple simulated agents. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11172 of *Lecture Notes in Computer Science*, pages 213–224. Springer International Publishing, Cham, 2018.

[28] C. R. Reid, T. Latty, and M. Beekman. Making a trail: informed Argentine ants lead colony to the best food by U-turning coupled with enhanced pheromone laying. *Animal behaviour*, 84(6):1579–1587, 2012.

[29] A. Reina, A. Cope, E. Nikolaidis, J. Marshall, and C. Sabo. ARK: Augmented reality for Kilobots. *IEEE Robotics and Automation Letters*, 2(3):1755–1761, 2017.

[30] V. Sperati, V. Trianni, and S. Nolfi. Self-organised path formation in a swarm of robots. *Swarm Intelligence*, 5:97–119, 06 2011.

[31] 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, 2020.

[32] V. Strobel and M. Dorigo. Blockchain technology for robot swarms: A shared knowledge and reputation management system for collective estimation. In M. Dorigo, M. Birattari, C. Blum, A. L. Christensen, A. Reina, and V. Trianni, editors, *Swarm Intelligence – Proceedings of ANTS 2018 – Eleventh International Conference*, volume 11172 of *LNCS*, pages 425–426, Cham, Switzerland, 2018. Springer.

[33] D. Sumpter and S. Pratt. A modelling framework for understanding social insect foraging. *Behavioral ecology and sociobiology*, 53(3):131–144, 2003.

[34] P. Szilágyi. EIP-225: Clique proof-of-authority consensus protocol. `https://github.com/ethereum/EIPs/issues/225`, 2017. (Accessed on 06/02/2022).

[35] M. S. Talamali, T. Bose, M. Haire, X. Xu, J. A. R. Marshall, and A. Reina. Sophisticated collective foraging with minimalist agents: a swarm robotics test. *Swarm intelligence*, 14(1):25–56, 2019.

[36] P. Turchin. *Quantitative analysis of movement: measuring and modeling population redistribution in animals and plants*. Sinauer Associates, 1998.

[37] G. Valentini, A. Antoun, M. Trabattoni, B. Wiandt, Y. Tamura, E. Hocquard, V. Trianni, and M. Dorigo. Kilogrid: a novel experimental environment for the Kilobot robot. *Swarm intelligence*, 12(3):245–266, 2018.

[38] G. Viswanathan, V. Afanasyev, S. V. Buldyrev, S. Havlin, M. da Luz, E. Raposo, and H. Stanley. Lévy flights in random searches. *Physica A: Statistical Mechanics and its Applications*, 282(1):1–12, 2000.

[39] N. W. Watkins, A. M. Edwards, V. Afanasyev, E. P. Raposo, H. E. Stanley, R. A. Phillips, E. J. Murphy, S. V. Buldyrev, G. M. Viswanathan, M. G. E. da Luz, and M. P. Freeman. Revisiting Lévy flight search patterns of wandering albatrosses, bumblebees and deer. *Nature*, 449(7165):1044–1048, 2007.

[40] E. O. Wilson. Chemical communication among workers of the fire ant *Solenopsis saevissima* (Fr. Smith) 1. the organization of mass-foraging. *Animal behaviour*, 10(1):134–147, 1962.