Università
Ca'Foscari
Venezia

Master's Degree programme — Second Cycle
(D.M. 270/2004)
in Informatica — Computer Science

Final Thesis

—

Ca' Foscari
Dorsoduro 3246
30123 Venezia

From Theory to Practice: Solving the Rendezvous Problem using Real Robots

Supervisor

Prof. Flaminia Luccio

Candidate

Davide Moro
Matriculation number 837188

Academic Year

2015/2016

Abstract

The increasing production of general-purpose, autonomous, mobile robots is a growing phenomenon that will strongly affect our daily life. These kind of robots are mainly studied in robotic and artificial intelligence fields. Only recently, studying the coordination of a team of robots has become popular also in the distributed computing area, where the researchers analyze the relationship between the solvability of a problem and the computational capabilities of the robots. Many works in this field address the *rendezvous* problem, i.e., the problem of grouping at the same point a team of robots initially placed in different positions within finite time. Nowadays, the results obtained in literature are only theoretical: real robots have been used only in few empirical works.

In this thesis we formally study how a team of *honest* robots moving on a ring graph can achieve the rendezvous, despite the presence of one *malicious* robot, that tries to prevent this from happening. We devise efficient algorithms for the oriented and the unoriented ring topologies.

Then, we map our theoretical model into a practical scenario and we apply our protocols using LEGO Mindstorms EV3 robots. We focus on the problems that need to be taken into account working with real robots and we then discuss how to overcome them and the technical limitations of our results.

Acknowledgments

I am grateful to my supervisor, Prof. Flaminia Luccio, for her continuous support, her prompt answers to my questions and the advice she provided.
A thank also to Marco Squarcina, for helping me in the practice section of this thesis.
Finally, I would like to thank my parents, Ivana and Giampaolo, for their constant support and their patience.

Contents

Introduction

In the last decade the production of small, cheap, generic, mobile robots has widely increased. This trend has motivated research in robotic and automation and in the *swarm robotic* field, the branch of artificial intelligence that studies the cooperation of simple autonomous robots in a group, taking inspiration from biology [11, 36, 43, 45].
Recently, the study of autonomous mobile robots has gained the attention in distributed computing area, where given a general *mobile robots model* the researchers investigate the relationship between the solvability of a problem and the capabilities of the robots. Many previous works on distributed algorithms focused on the *mobile agents model*, where the agents are software moving in a network. These two models share many common points given that in both cases the agents/robots move autonomously, and many computational problems are analyzed in both the models. Among these problems, one of the most studied is the *rendezvous* [40], or *gathering*, namely the capability of a team of robots initially placed in distinct positions, to meet at a same point, without having a complete knowledge of the environment in which they are moving.
Nowadays the theory is quite far from the practice. The majority of the works have never been tested in a real scenario [3, 15–18, 33]. Conversely, the few works that take advantage of programming or real robots simulations are quite empirical and lack of a solid theoretical basis [12, 38]. Unlike these works, in this thesis first we propose novel theoretical solutions for the rendezvous problem, and then we verify that our solutions also work with real robots.

The contributions of this thesis are summarized in the following. We discuss the most recent works related to the rendezvous problem in the mobile agent and in the mobile robot models.
We then extend the model of [24] to the case of synchronous agents. We propose a rendezvous algorithm for the oriented ring network, with $k \geq 2$ honest agents and one malicious agent. We prove an impossibility result for

the unoriented ring and we propose two novel algorithms for this case: one algorithm for two honest agents, and one for $k > 2$ agents.
We execute our protocol with the general-purpose LEGO Mindstorms EV3 robots, focusing on the devices that have been used to implement the robot capabilities, discussing the problem of their general unreliability and proposing some solutions.
To the best of our knowledge, this is the first time that the rendezvous is tested with one malicious robot that does not cooperate with the other. We think that, with the increasing of autonomous mobile robots, the presence of an intruder robot will be a novel topic to study in security and distributed areas.

The thesis is organized as described in the following. In Chapter 1 we introduce the notation related to the mobile agents and the mobile robots models and the major problems studied in this area. In Chapter 2 we summarize the works related to the rendezvous problem in the mobile agent model, in particular the ones studying the problem in presence of some faults. In Chapter 3 we discuss the works concerning the rendezvous problem in the mobile robot model, focusing on the ones in which the robots have some physical limitations and the ones which have been practically simulated. In Chapter 4 we extend the model of [24] to the case of synchronous agents and we propose some novel algorithms to solve the rendezvous in the ring. In Chapter 5 we implement in practice the proposed protocols using real robots, outlining the technical limitations we faced. In Chapter 6 we conclude with the final remarks and the suggestions for future works.

Chapter 1

Model and Definitions

In this chapter we introduce the mobile agents and the mobile robots models and the notation used in the thesis. The mobile agent model is defined in the first section, while its advantages and the problems mostly studied in this model are summarized in the second section. In the third section, we introduce the security issues of the mobile agents model, discussing two well-known problems: the *black hole* and the *intruder capture*. In the last section, we give a formal definition of the different mobile robots models used in literature.

1.1 Mobile Agents Model

Mobile agents were initially proposed in the area of software engineering and are nowadays applied in many fields of distributed systems. A *mobile agent* is a software executed on a computer of a network, that can migrate from a host to another maintaining its execution code and its data. An agent can exploit the host in which it is located and can communicate with the other agents at the same host, with the aim of performing a task [42].

We now describe a general mobile agents model, used in literature (see e.g., [21]). This model is refined and adapted to various applications, specifically, the knowledge of each agent related to the network topology and the method of communication of agents could change substantially.
A mobile agents system is modeled as:

- a connected graph $G = (V, E)$ where V is the set of hosts, and E the set of links between them. G is undirected, i.e., an agent can move on

an edge (u, v) from u to v, and vice-versa.

- a set of mobile agents A.
- a port numbering bijective function δ,
  $\forall u \in V\ \delta_u : \{(u, v) \in E\} \rightarrow \{1, 2, \ldots, deg(u)\}$
  that numbers each port of a node u in a different way, allowing agents to distinguish the ports of the node.
- a graph labeling function, $\lambda : \{v \in V\} \rightarrow \{1, 2, \ldots, |V|\}$, that labels each node with a number.
- an agents labeling function, $\pi : \{a \in A\} \rightarrow \{1, 2, \ldots, |A|\}$, that labels each agents with a number.

We have an *anonymous graph* if all the nodes have the same identifier, in this case the λ function can be omitted from the model. Likewise, two or more agents are *anonymous agents* when they have the same label, if all the agents are anonymous the π function can be omitted.
From here on in we use the notation $n = |V|$, $m = |E|$ and $k = |A|$.
A mobile agents algorithm assigns to each agent a position on the graph according to a specific function $p : A \rightarrow \{1, 2, \ldots, |V|\}$. We refer to the memory of an agent and of an host with the generic term *state*. Each agent can be seen as an automaton that starts with a particular state s_0 of a finite number of states S. An agent can remain stopped or move from the node where it is located to one adjacent node. It can read and possibly modify the state of the node where it is located. It can modify its current state and communicate with the other agents at the same node. The memory of the agents can be considered infinite or, in more realistic models, it is finite and typically a function of n.

The models we will face refine the general one, and typically one of the following *communication techniques* is adopted:

- *Face-to-face* communication: an agent located at a node can communicate with all the other agents at the same node.
- *Whiteboard*: an agent accesses in mutual exclusion to the state of a node to perform a read/write operation.
- *Token*, or *pebble*: it is a weaker variant of the whiteboard communication. An agent is equipped with one or more tokens. All tokens are indistinguishable. An agent can place a token at a node (or at a port, it depends on the model) to mark its passage. It can also pick a token when it found it.

Agents can be *asynchronous*, i.e., they move from a node to a neighboring one in a finite but unpredictable amount of time, thus this information can not be exploited to synchronize agents during the execution of a protocol. This is justified by the fact that in real scenarios we can not rely upon the time needed to send data trough a network connection. Conversely, the agents are called *synchronous* if they move from a node to a neighboring one in a finite fixed amount of time.

Compared to other distributed system models, mobile agents one represents an alternative to the message passing model. The equivalence of the two models has been proved in [13].

1.2 Advantages and Basic Tasks

A distributed protocol based on a mobile agent is easier to maintain with respect to a typical client-server application. The former case requires only the code for the software agent, while the latter needs one program running on the client and another running on the server.
It may happen that a client-server application requires many communications between the client and the server. In this case a mobile agents protocols would reduce the network traffic, since an agent, once moved from an host to another, makes local computations on the host, without requiring further network connections. However, the mobile agents solution could increase the bandwidth required, if the dimension of a mobile agent is much greater than the data sent in a client-server application [42].

Although none of the existing programming languages can implement a mobile agents protocol, this topic is theoretically interesting. Regardless of the task of a mobile agents protocol, some problems are recurring and they have been intensively studied in literature. We briefly list the main ones (see e.g., [4, 9, 10, 26, 30]).

- *wakeup*: suppose that more agents have to perform a task, but only some agents are currently active. The wakeup problem consists of using active agents to notify a signal to inactive ones, so that the inactive agents can became active.
- *traversal*: it is the problem of moving each agent in such a way it visits all the nodes of a network, possibly only once, and then it returns to the starting node. It can be used when an agent has to collect data, or search or filter some information.

- *election*: it is the problem of choosing exactly one agent, called *leader*, among all the agents. The leader could be used, for example, to coordinate all the other agents, called *followers*. The election problem is more difficult when agents are anonymous and their labels cannot be exploited to distinguish the agents.
- *rendezvous* or *gathering*: it is the problem of grouping all the agents that start from different positions at a same node. This is applied for example when agents have to move together or share the data. For its importance is one of the most studied problem in literature and it is the one we will analyse in this thesis.

1.3 Security issues

In the previous section we assume that the agents cooperate to achieve a goal, trust each other and the host on which they are located. These hypotheses are not always realistic as they do not consider the security issues strictly connected to the distribute systems.
A first kind of problem, known as *harmful agent*, is the presence of a malicious mobile agent that interferes with the execution of the other agents, the honest ones, and it can even prevent them from fulfilling their task. When a malicious agent behaves as the honest ones with the intent to mislead them, it is called *Byzantine*. Furthermore, there could be more than one malicious agent.
A harmful agent can represent, in environments like the Internet, a virus transmitted trough e-mails.
Another problem, called *harmful host*, arises when an host is untrustworthy. If an host is affected by a virus this can be seen as a particular case of the previous one, where a stationary malicious agent remains at an host. A harmful host could represent also the case of a reliable network where there is a software or hardware error inside an host, impacting the behavior of agents.
Unexpected situations occur also when transmissions fail because of some error, like for example a temporary loss of connection. Even communication system could be subjected to errors. When agent are mobile robots, the sensors they used to implement their capabilities are not fully reliable. It follows that even malfunctions of an honest robot could affect the correctness of an algorithm.

If basics tasks such as rendezvous and election in a fault-free setting, have

been intensively studied in literature, the design of fault-tolerant protocol is still an open context. Nowadays the research focuses on studying under what conditions these problems are solvable in presence of a specific fault. The major effort concerns the investigation of the minimal hypotheses for which a problem can be solved. The quality of a proposed protocol is determined in terms of the number of moves in the worst case and, for synchronous settings, also by the worst case time required. We briefly summarize two well studied cases of fault-tolerant problems, a complete overview of these topics can be found in [31].

1.3.1 Black Hole Problem

The first type of fault we present is a stationary malicious agent that remains at an host and it is so powerful that it kills any other honest agents entering at the same host. This is clearly a harmful host problem that for its particular behavior is called *black hole*.

Definition 1.3.1. *Given a team of honest agents starting from a same node, called home base, the* ***black hole search*** *is the problem of exploring a graph with a black hole, in such a way that at least one agent survives and all the surviving agents know where the black hole is [31].*

The black hole search can be solved with an algorithm called *cautious walk*, where agents exploit the whiteboards of the nodes. At the beginning any port of a node is marked as *unexplored*. When an agent starts moving on an edge mark a port as *dangerous*, because it can not know if this edge leads to the black hole. After that an agent crossed a safe edge it traverses it again in opposite direction and mark the relative port that was dangerous as *explored*. A node is considered *expanded* if all its ports are explored. In cautious walk, as the name suggests, an agent moves carefully: it avoids to move on an edge if the relative port is dangerous and each time that a node becomes expanded it returns to the home base and notify this fact. Under the hypothesis that agents know n and the maximum degree d of a node, $d+1$ agents performing cautious walk protocol are necessary and sufficient to solve the black hole search in any graph topology. It is intuitive to see that d agents could fail because if the black hole has degree d all of them will be killed. Cautious walk has a cost of $O(n^2)$ moves on a n-node graph.
If agents have sense of direction, i.e., they know if two different edges lead to a same node, just two agents suffice for solving the black hole search, with the previous cost $O(n^2)$ moves [31].

If the agents have the full knowledge of the graph structure, two agents suffice but in this case the complexity can be reduced to $O(nlog(n))$ moves [31].

1.3.2 Intruder Capture Problem

We now describe a harmful agent problem, called *intruder capture*, where a team of honest agents has to protect a network from a malicious agent. Honest agents cannot move at a node where there is the malicious agent, and vice-versa. A node occupied by an honest agent is *protected* or *clean*, a node becomes *contaminated* if it is occupied by the malicious agent or if it has a neighbor contaminated. The intruder capture problem considers that all the network is contaminated except one node, the home base, occupied by all the honest agents and these ones have to apply a sequence of movements for decontaminating the entire network. Honest agents use a face-to-face communication.
Some of the research focus on finding *monotone decontamination strategies*, i.e protocols for which, when a node is cleaned, it will remain cleaned until the end of the decontamination. The quality of the protocols is measured based on the minimum number of honest agents required, on the total number of movements performed and on the total execution time of the decontamination.
Unlike the black hole problem, the intruder capture cannot be generalized for any topology and the problem of finding the optimal number of honest agents in an arbitrary graph is NP complete [31]. It follows that the works on this topic are related to specific topologies such as trees, hypercubes, tori. Even for standard topologies the optimality bounds of number of agents, time and moves have not always been proven and in many cases they remain open questions. Furthermore it is interesting to study how increasing the honest agents capabilities affects the current results. In many studies, for example, the honest agents have the visibility power, namely an agent at a node see if there are other honest agents in an adjacent node.

Black hole and intruder capture are two well known problems that do not cover all the possible security aspects related to the mobile agents concept. Real scenario remains nowadays too difficult to be modeled because more than one problem could arise on a network, even of different nature, and besides, a problem could change over the time. We have to consider also that networks communications are intrinsically unreliable. Although the state of the art is far from modeling protocols that can be safely employed in real situations, lots of efforts had been done in this direction. The trend is

to model problems more an more complicated, and in the next chapter we will analyse some recent studies that in particular refer to the rendezvous problem.

1.4 Mobile Robots Model

The typical mobile robots model used in literature is the *Look-Compute-Move* model. This model can have many variants related to the features of the robots, (see [40] for a survey).
A *robot* is a computational unit (the terms *agent* or *bot* can be used as well, we use the term robot to distinguish this model from the mobile agents one). Let $R = \{r_1, r_2, \ldots, r_n\}$ be a set of n robots over a two dimensional plane. The position of a robot r_i at time t is denoted as $r_i(t)$. Each robot executes cyclically these three actions:

- *Look:* the robot obtains the coordinates of the other robots in its visibility range with respect to its own coordinate system.
- *Compute:* the robot uses an algorithm A, equal for all the robots, that takes in input the coordinates obtained in the Look phase and it returns in output a destination point.
- *Move:* the robot moves towards the destination point computed in the previous phase.

A robot is considered *autonomous*, not coordinated by a central unit such as a server; *anonymous*, indistinguishable by the other ones and *silent*, not being able to communicate directly with the other robots because any decision must be taken only according to the other robots positions.

Synchronization Model
A robot is *active* if it is performing one Look-Compute-Move cycle, otherwise it is *idle*. The time required by each phase of one cycle and the activation/idling phase of each robot can be modeled in many ways, the three employed in literature are the following.

- **ASYNC**. The robots move asynchronously. Each robot executes a Look-Compute-Move cycle at an unpredictable instant and the time required by each phase is not known in advance. Furthermore a robot may not reach its destination point. The only two assumptions, one related to space the other related to time, are the following. If a robot does not reach its destination point, it moved at least for a minimum

constant distance δ, and each phase of a cycle takes at least a minimum constant time ϵ. These few assumptions make this setting difficult to addressed. In fact when a robot takes a *snapshot* of the position of the other robots, during the elapsed time between the Look and the Move phase of the robot may the positions of the other robots change and so the robot moves according to an inconsistent situation. Moreover a robot cannot distinguish among the other robots which one is moving and which one is stopped.

- **SSYNC**. The Look, Compute, Move phases are synchronized with a global clock. At each cycle some robots, but not all, could be idle. The problem of *ASYNC* timing is no more valid, in fact a robot sees only stopped robots and starts moving at the same time of the others. Given the positions of the robots at time t, their positions at time $t+1$ can be precisely obtained. The time taken to move is not an important assumption, hence by convection it is considered null and the robots are called *instantaneous*.

- **FSYNC**. The robot are fully synchronized. It is a special case of the *SSYNC* model in which all the robots are active in any round.

ASYNC setting is the most difficult to deal with and the problems solvable in this setting are always solvable in the *SSYNC* and *FSYNC* settings. Conversely the fully synchronized is the simplest one and there are problems solvable in *FSYNC* that have been proved to be unsolvable in *SSYNC* and by consequence in *ASYNC*. That is the case of the rendezvous problem with certain conditions, discussed extensively in Chapter 3.

Robot Features

The computational capabilities of a robot could vary in every model, we summarize the different assumptions that can be made about, following [40]

- **Visibility**. A robot can a have an *unlimited visibility* if it is able to see all the other robots in the plane, or a *limited visibility* if it sees only up to a constant distance.

- **Coordinates system**. Each robot has it own compass centered to its position denoted by two coordinate axes x and y that have a *direction*, i.e., inclination of the axes with respect to the plane, and an *orientation*, i.e., positive and negative sides. Different systems of global coordinates agreement between the robots may be employed. The robots have a *consistent compass* if they agree on both the orientation and direction of the axes. When they agree only on the orientation and direction of one axis the system is called *one axis*. The *chirality* is the system in

which the robots agree only on the axes orientation but not on their directions. Finally if the robots do not agree neither on the direction nor on the orientation of the axes the system is called *unoriented*.

- **Memory**. A robot is called *oblivious* if it loses all its information at the end of any Look-Compute-Move cycle. Conversely, a *non-oblivious* robot maintains the data computed during a previous cycle. In this last case the memory can be considered *unbounded*, if it is potentially infinite, or *limited* if it is finite.
  The use of oblivious robots may is questionable because in the practice even the cheapest devices are equipped with a memory. The memory could be exploited to store the information computed during the algorithm execution. Actually the majority of the theoretical works assume that robots are oblivious for the following reasons. First, this choice enhances the *self-stabilization* of the protocol, i.e., the possibility to terminate regardless the initial state of the robots because they have not a state, and the *fault-tolerance* of the algorithm, in this case related to possibly faults of the memory state. Furthermore, a protocol for oblivious robots can also be executed by non-oblivious robots, while the contrary it is not always true.

- **Shape**. The first works on mobile robots assumed the robots are dimensionless object, i.e., points. Recently more realistic models used robots that occupy a space. These robots called *fat*, or *solid*, robots are disks with an equal diameter. They cannot overlap, but only touch and when this happens they stopped. The robots can obstruct the view of another one or not, in the last case they are called *transparent*.

Chapter 2

Rendezvous in the Mobile Agents Model

In this chapter we describe some of the works related to the rendezvous problem in presence of faults in the mobile agents model. The first section is about the delay faults in the asynchronous and synchronous settings. In the second section we discuss how the agents can gather if the communication mechanism is not reliable, specifically referring to the problem of faulty tokens. The last section concerns the presence of one or more harmful agents on a network that hinder the honest agents.

2.1 Rendezvous in presence of delays

In this section we describe the works [14, 19] regarding the rendezvous problem of two mobile agents in a general graph. In the former the agents are subjected to an unpredictable delay, in practice they move asynchronously. In the latter the agents move synchronously but they are affected by a delay that arise according to a particular distribution.

2.1.1 Asynchronous Setting

In the work of Czyzowicz et al. [19] the authors give a solution for the rendezvous problem both in an anonymous connected graph without knowing its size and also in a plane. We discuss only the results for the graphs since the execution in a plane will be analyzed in the next chapter. In this model

two agents that move asynchronously have to meet without using tokens. There is no a particular malicious agent but the adversary is the responsible of the asynchronous moves of the agents. In particular not only it can delay the agents but in can also move them back and forth within a same edge. This last point, that can be neglected if the agents meet at a node, is actually important because agents, despite the standard mobile agent model, could meet also inside an edge. The relaxation of the rendezvous problem is due to the fact that otherwise the problem remains unsolvable even for trivial topologies, such as a graph with only two nodes. The two agents have distinct integer labels but each agent does not know the label of the other one.

The authors propose a solution that forces the two agents to move in opposite directions along a same sequence of edges, called *tunnel*, in such a way that eventually they will meet, regardless the delay they have. This is obtained with a combinatorial algorithm. Both the agents rely on a function φ that given an integer returns a quadruple (i, j, s', s'') where i and j are two natural integers and s' and s'' two sequences of positive integers of the same length. The two agents a_1 and a_2 execute a procedure that considers for an increasing integer k its relative quadruple $\varphi(k)$. When the label l of an agent coincides with the value i or j of the enumeration returned, the agent chooses a deterministic path, i.e., a sequence of ports in the graph. When the value of k returns exactly the sequence (l_1, l_2, s', s'') with l_1 and l_2 the labels of the agents a_1 and a_2 respectively, s' a path from the starting node of the agent a_1 to the agent a_2 and s'' the reverse path, the two agents rendezvous because both are moving in a tunnel.

The complexity depends on the number of enumerations tested before finding the right one but a priori it is impossible to devise a function that returns quickly the correct enumeration since neither the topology nor a bound on its size are given as input. Even if the algorithm is deterministic, it is evident that its complexity is exponential in the labels of the agents, because it tests any possible pair of integers, and in the distance of the two agents in the starting configuration because it tests any sequence of numbers between two possible nodes. Although the theoretical result is infeasible in the practice, this is very interesting because it solves a problem that had been for a long time open [25]. It could be a starting point for new research in this topic, in fact a polynomial algorithm for the asynchronous rendezvous in a connected anonymous graph has not been devised yet and its existence is nowadays and open question.

2.1.2 Synchronous Setting

Chalopin et al. [14] study the rendezvous in presence of delay faults. In their model two synchronous agents have to meet at a node. In a round an agent may move or not but if a fault occurs the agent remains at its current node for the entire time of the round, even if it wants to move. The two agents start moving in possibly different rounds, they have a potentially infinite memory and they are not equipping with tokens for marking nodes. They are labeled with two distinct positive integers but each agent knows only its own label. If the last constraint does not hold it is easy to develop a rendezvous algorithm: the agent with the smallest label remains stopped waiting the agent with the greatest label that is moving.
The authors analyze the conditions for the rendezvous in an arbitrary graph, provided that the agents do not know its size. They concentrate on three kinds of fault distributions: *random*, *bounded* and *unbounded*. In the random distribution a fault can occur with constant probability p independent in each round and for each agent. In the bounded distribution a fault can occur up to c consecutive times. In the unbounded distribution a fault can occur for any finite number of consecutive times.

For the random fault distribution the authors noticed that this setting is a particular case of the asynchronous one where the time to traverse an edge is no more constant but varying according to the number of times that an agent has been delayed, i.e., it is a multiple of the unit of time t. The solution for this problem is in fact a refined version of a previous work for the asynchronous case [29]. The synchronous version of the protocol is not obtained straightforwardly from the asynchronous one because in that case the agents can meet also inside a node while in the model proposed this is forbidden. The agents move in such a way that any time the original algorithm converges with a meeting inside an edge, in the refined algorithm they meet at a node. This solution has a polynomial cost in term of the number of moves, with respect to n and logarithmic with respect to the greatest label L among the two labels. Note that in all the protocols of the article the agents move according to their labels, and so the complexities of these algorithms also depend on these parameters.
In the bounded fault distribution, the agents do not know c, the maximum number of consecutive times that a fault can occur. If they know c, they may run any synchronous algorithm for the non-fault setting, trying many times every move, according to the maximum delay bound, in order to synchronize with the other agent. The obtained algorithm has the same cost in term of moves of the non-fault one, but with a greater execution time. The authors

devise an algorithm where the parameter c is estimated by the agents during the execution. The algorithm has a cost polynomial in n and logarithmic in c and in the label L. The execution time is instead polynomial in n, c and L. The time complexity is greater than the moving complexity because one agent spends a lot of time remaining idle while the other is possibly moving, to avoid the two agents cross on an edge.
The most critical type of fault, the unbounded one, negatively affects the possibility of perform rendezvous. The authors show that in this case rendezvous is impossible for a general graph without knowing its size. In fact, they prove that the problem is unsolvable in the ring if n is unknown because, due to its symmetry, there are situations in which the two agents continue to cross but they never meet at a node. The problem is in particular related to the cycles contained in a graph and so as a natural consequence the authors check if an arbitrary connected acyclic graph, i.e., a tree, can be solved without knowing n. They first start by solving the rendezvous on an oriented ring with the size n known and then they extended this result to an arbitrary tree.
The case of an oriented ring can be solved by moving each agent with id i in the same direction for $2ni$ times, achieving a cost of $O(nl)$ with l the smallest label.
Interestingly, the case of a generic tree can be lead back to the one of a ring if the agents, exploiting the ports numbering of the tree, visit the nodes of the tree in a fixed order given by a walk, called *basic walk*. In a basic walk an agent that starts at a node v visits each edge of a tree of size n exactly once and after n steps it returns at v. The basic walk works as follows: leave v from the port 0, arriving at any other node from a port p leave it from the port $(p+1)$ mod d with d the degree of the node. The agent realizes to be again at its starting node maintaining a list of the ports from which it has already entered or exited and so it learns the size n of the tree. The tree visited in the basic walk order can be seen as an oriented ring of size $2(n-1)$ and thus, according to the previous result, the rendezvous in a tree is solved by moving an agent with label i for $2(2(n-1))i$ times with the previous cost $O(nl)$.

The main results of the article are summarized in Table 2.1. The authors devise a rendezvous algorithm for random distribution fault with cost polynomial in n and logarithmic in L. For the bounded fault distribution the cost is polynomial in n and logarithmic in c and L. The execution time is instead polynomial in all these three variables and obtaining a solution that works faster remains an open problem. They show that without knowing the size of the graph the rendezvous is not solvable in the unbounded fault distribution

case, but it can be solved in the class of trees with a cost of $O(nl)$ moves.

Type of delay	**Cost (moves)**
random with probability p	$O(P(n) + log(L))$
bounded up to c consecutive times	$O(P(n) + log(c) + log(L))$
unbounded	impossible for general graph $O(nl)$ for tree graphs

The size of the graph n is unknown, l and L are respectively the smallest and the greatest label, P is a polynomial time algorithm.

Table 2.1: Rendezvous of two agents subjected to delay faults [14].

In the conclusion the authors explain an algorithm to solve rendezvous in presence of unbounded delay faults given an upper bound q on the size of the graph. The proposed solution has a polynomial cost in q but exponential in l, the minimum label. The existence of an algorithm for such kind of problem with a polynomial cost both in q and in l is nowadays an open question.

2.2 Rendezvous when Tokens Fails

We describe how faulty tokens can influence the solution of the rendezvous problem [22]. In this article a token can disappear at any time and never appear again. In previous works a token disappears but only immediately after that an agent leaves it at a node [20, 32]. For the case of *Byzantine* tokens that can disappear and reappear many times see [27].

Das et al. [22] faced the problem of rendezvous when the agents are equipped with a token that can fail anytime. They shown that for a ring graph and for a general graph the solvability conditions are the same as the non-faulty setting.
In their model every agent is equipped with a token that is indistinguishable from the other ones. An agent can leave a token at a node to mark it and sign its passage. Two or more agents can also apply a face-to-face communication mechanism when they are at the same node. In [20] the author proves that the agents must know the number of agents, k, in order to achieve rendezvous. In the proposed model the agents know both k and the size of the graph, n, they move asynchronously and start from different nodes. A token is not

reliable but it can disappear at any moment once for all. The number of tokens that can fail, f, is $0 \leq f < k$. In practice the proposed solution works despite the number of tokens that fail, except in the case that all the tokens fail but in this case this must be detected by the agents. The fact that no token can fail, i.e., $f = 0$, is important because, as the authors claimed, knowing that at least one token fails could be used as a symmetry-breaking technique. Hence obtaining a protocol working both for the faulty and for the non-faulty setting was not a trivial challenge.

The first problem studied is the rendezvous in a ring, then this result was extended to a general graph. In any case when the problem is not solvable an agent realizes it and terminates the protocol in a failure state. The authors first analyze the solvability conditions for the non-faulty scenario and then they propose some solutions that work at the same conditions and with the same costs.
In the case of ring the rendezvous problem of asynchronous agents, given n and k, is solvable according to the initial disposition of the agents in the ring. Some configurations in fact, due to their symmetry, make it impossible to achieve rendezvous. Regardless the precise definition of such kind of configurations it is intuitive to see that any agent knowing n and k can get the starting configuration of all the agents. For each agent it is enough to leave the token at its starting node and then to move for n steps, so that it can find all the other tokens. Another important aspect is that from an initial configuration S that is solvable, it is possible to deterministically obtain one node, called *RV-point(S)*, where the agents meet. The *RV-point(S)* is always one node where an agent started.
The previous considerations can be exploited also in the faulty setting as proposed in the article and briefly summarized in the following. After that an agent computes a tour of the ring, if it does not find its token, it continues to move in order to notify all the other agents its starting position, because the token could disappear before another agent finds it. Otherwise, if an agent finds again its token, it remains at its starting position. To prevent the situation in which all the agents stop, because no fail occurs, the agent at the *RV-point(S)* is elected as leader and move anyway. The moving agents continue to turn until all the agents positions are collected and notified to the stopping agent. At this point the agents move at the *RV-point(S)* node following the leader.
This protocol has the same cost of the non faulty setting: $O(kn)$ total moves where k and n are respectively the number of agents and the size of the graph.
The solvability condition of a generic graph is again related to the starting

configuration of the agents. Suppose that the graph G is *bi-colored*, the *black* nodes represents the starting nodes of the agents, the *white* nodes all the others.

Definition 2.2.1. *The **bi-colored view** of a node v in a graph G is the infinite tree rooted in v, that has for children the neighbors node of v in G. Each child u is colored as in G and it is recursively the root of the bi-colored view of u.*

In a generic graph G the rendezvous problem is solvable if and only if each node of G has a distinct bi-colored view [9]. A bi-colored view is infinite but in a graph of size n, two bi-colored views are equal if and only if up to depth $n-1$ they are equal, hence here and on we consider the bi-colored views limited to depth $n-1$. Once computed the bi-colored views it is possible to build a graph H of G where all the vertices with a same bi-colored view are merged into a unique vertex. A condition to solve the rendezvous equivalent to the previous one is that the graph H is isomorphic to G. In this case, by ordering the bi-colored views in some deterministic order it is possible to compute a single node, called *RV-location(H)*.
The protocol to solve rendezvous in a generic graph exploits the theoretical results described above and works as summarized in the following. An agent leaves its token at its starting point v, that is black and start visiting the nodes according to the bi-colored view of v. During a bi-colored view exploration some node is visited more than once and it can be of an inconsistent color, in fact a white node may become black because some token disappeared. For this reason the agent applies twice the bi-colored view exploration and only if it obtains two times the same bi-colored view this one is considered consistent. Similarly to the case of ring, an agent that found an inconsistent bi-colored view in which its starting node is no more marked by the token continues to move to notify the other agents its starting position. When all the agent know the exactly bi-colored view of their starting nodes, each one can compute the graph H and know if the configuration is solvable or not, and in this last case it exits the protocol with a failure. As in the previous case, when the problem is solvable it is not possible that all the agents remain stopped because the agent at the *RV-location(H)* becomes the leader and moves collecting all the other agents at the *RV-location(H)*.
The depth first visit up to $n-1$ levels requires $O(D^{2n})$ moves for each agent, with D the maximum degree of a node, and so the protocol has a complexity of $O(kD^{2n})$ total moves.

Table 2.2 summarizes the results obtained in the article. Two protocols one for the rings with cost $O(kn)$ and another for a general graph with

Graph	Cost (moves)
ring	$O(kn)$
general	$O(kD^{2n})$

The size of the graph n and the number of agents k are known, D is the maximum degree of a node.

Table 2.2: Rendezvous complexities when tokens fail [22].

cost $O(kD^{2n})$ had been proposed. In both the two cases the solvability conditions and the costs are equal to the rendezvous problem without faults. An improvement of the exponential complexity of the unknown graph case cannot be achieved, unless it exists a more efficient way to explore or traverse a graph.

2.3 Rendezvous despite a Harmful Agent

In this section we deal the problems presented in [24, 28]. In these works one or more harmful agents move through a network hindering a group of honest agents that try to achieve rendezvous. The former considers there are more *Byzantine* agents that can mimic the behavior of the honest ones to mislead them. The latter studies the problem in presence of one *Malicious* agent that cannot impersonate an honest agent but it can block it.

2.3.1 Byzantine Agent

Dieudonné et al. [28] analyze the rendezvous problem when up to f agents are Byzantine. They study this problem for a general graph, with synchronous agents identified by distinct labels. They assume the agents may wake up at different rounds and each time that an active agent meets a sleeping one, the sleeping agent is woken up. In this model all the agents know the upper bound f of the maximum number of Byzantine agents. The agents can apply only a face-to-face communication. The authors define as *f-Byzantine gathering* the rendezvous problem with termination, i.e., all the honest agents know when the rendezvous happens, under the previous hypotheses and provided that at least one honest agent is awake in the starting configuration. At the end of a rendezvous algorithm the honest agents cannot know exactly the number of

honest agents because some Byzantine agent could maintain always a correct behavior. They cannot discover n either, given that some Byzantine agent could escape from the honest agents during all the algorithm execution.
The authors studied this problem with two types of Byzantine agents: a *weakly Byzantine* agent that can mimic the honest agent behavior but it has a fixed label, and a *strongly Byzantine* agent that can also change its label with an arbitrary one, possibly the label of some honest agent. Both these problems are examined under the condition that n is known or unknown. The article focuses on the minimum number of honest agents necessary to solve the *f-Byzantine gathering*.

The proposed algorithms exploits previous results obtained in literature to explore a graph and to perform rendezvous without malicious entities. The weakly Byzantine problem is a generalization of the rendezvous procedure for 2 agents using tokens of [34] that takes the id of an agent as a parameter. The idea of the modified algorithm is that when two agents meet they follow this procedure using the lowest id among the two, and by induction two groups of agents achieve the rendezvous. The problem is that a weakly Byzantine agent could arbitrarily leave a group of agents, but since the rendezvous procedure is deterministic the other agents realize it is a Byzantine agent because it did not move correctly.
If the the size of the graph is known, each honest agent has an own blacklist of Byzantine agents, updated every time it detects a Byzantine agent and shared with the other agents when they meet. The previous procedure is applied only for the agents whose labels are not in a blacklist so that eventually the misleading behavior of the Byzantine agents become ineffective and the rendezvous is achieved.
When the size of the graph is not known, the solution is obtained with a variant of the previous one in which an agent memorizes also the list of the agents that are not in its blacklist.
For both the cases the algorithms proposed have a polynomial cost in term of moves. The former case does not require a minimum number of honest agent while in the latter case at least $f + 2$ honest agents are necessary. In both the cases the two bounds have been proved to be tight.
The data structure used to solve rendezvous in presence of weakly Byzantine agents cannot be applied to the case of strongly ones. In this case in fact the Byzantine agents can continuously change their labels an so it is impossible to distinguish a Byzantine agent with the label of an honest one from an honest agent. The key point of the proposed solutions for the strongly Byzantine case is meeting a group of agents such that at least $f+1$ agents are honest, called *(f+1)-tower*, so that another agent can safely join this group,

until all the honest agents meet. The number of honest agents necessary to grant the formation of a *(f+1)-tower* is $2f + 1$ because all the f Byzantine agents could leave a group. The authors start with a randomized solution for the case of n known. The idea of this solution is to apply a random walk such that a *(f+1)-tower* is obtained with an high probability in a polynomial number of moves.
The deterministic solution for n known implies that all the agents moves knowing the starting positions of the agents. This information is not available in the model and it requires that the agents enumerate all the possible configurations and test them one by one. When the right configuration is chosen the formation of a *(f+1)-tower* is guaranteed, but this could happen also previously.
If n is not known the agents test the starting configuration for every possible value of n. With respect to the previous case the use of a token is essential to explore the graph but the model does not envisage that the agents have the tokens. To overcome this problem the role of token is played by a group of agents. In this manner the bound of necessary agents increases to $4f + 2$, one $f + 1$ tower of agents used as token and one $f + 1$ tower used as explorer.

	Weakly Byzantine	**Strongly Byzantine**
$\boldsymbol{n}$ **known**	Upper and lower bound: 1	Upper bound: $2f + 1$ Lower bound: $f + 1$
$\boldsymbol{n}$ **unknown**	Upper and lower bound: $f + 2$	Upper bound: $4f + 2$ Lower bound: $f + 1$

Table 2.3: Bounds of rendezvous in presence of up to f Byzantine agents [28].

The main results of the article are reported in Table 2.3. For the weakly Byzantine agents problem the authors propose two efficient algorithms with a tight bound on the number of honest agents required, while in presence of strongly Byzantine agents they propose two combinatorial algorithms with exponential complexities. In the last case they prove a lower bound of $f + 1$ necessary honest agents but they do not prove an upper bound on the number of sufficient honest agents, and so they do not know if the two bounds of the algorithms are tight.
Discovering whether the bounds for the strongly *f-Byzantine gathering* problem with n known and unknown are tight, as well as determining the existence of polynomial algorithms for these problems remain two open questions.

2.3.2 Malicious Agent

Das et al. [24] study the rendezvous problem despite a very powerful malicious agent that blocks the honest agents preventing them from meeting. In their model a malicious entity called M moves asynchronously and arbitrarily fast trough the network. It is omniscient with respect to the network topology, the number of honest agents and even their positions. The objective of M is preventing the rendezvous or trying to delay it as much as possible when it is unavoidable. The malicious agent cannot destroy an honest agent like a black hole, but unlike this one the malicious agent can choose to remain at a node or to move. M is always distinguishable by the honest agents unlike a Byzantine agent, but compared to this one the malicious agent has the stronger capability of blocking the other agents. From a practical point of view this scenario could model a network affected by a virus that cannot be deleted.
Honest agents are quite weak, they move asynchronously but respecting a FIFO order of the links, i.e., an agent cannot overtake another one in a same edge if both are moving in the same direction. The agents are anonymous, they do not know neither n nor k and they applied only a face-to-face communication. Their capabilities are enhanced to face some particular situations, as explained in the following. An agent can never move at a node occupied by M and vice-versa. To respect this constraint an honest agent is able to detect if M is located at an adjacent node.

Definition 2.3.1. *A displacement of honest agents on a graph G is called* ***separable*** *if there exists a connected component F of nodes without honest agents which, when removed, disconnects G and at least two honest agents are in different partitions of G.*

A separable initial configuration makes rendezvous unsolvable, in fact M can continue to move at the node of F blocking the two agents in the different partitions so that they never traverse F and consequently they never meet. The separable condition is not only problematic at the beginning of the algorithm but also during its execution, because when the agents move a such kind of configuration could possibly arise.
The authors study algorithms for the rings and the meshes. The rings do not present the problem of separable configuration but they cannot be trivially solved due to their high symmetry. The rendezvous problem is solvable only if the ring contains a special node, $o*$, that the agents distinguish from the other ones. In fact, supposing that all the agents go in the same direction, also M could move in this direction and, given that the agents do not know

n, they would continue to turn around if all the nodes of the ring were equal. Two types of ring have been studied in the article: the *oriented ring*, where the agents have a common sense of direction, i.e., they agree on the clockwise and counter-clockwise directions, or the *unoriented ring*, where there is no a common sense of direction. The latter is more involved and the authors showed it cannot be solved for any even value of k, even if k is known by agents. In effect, in this case some completely symmetric situations could arise and no deterministic algorithm could break the symmetry of agents allowing them to gather at a node.
The rendezvous in oriented rings is achieved by moving the agents as summarized in the following. An agent stops if it finds the special node $o*$ and it reverses its direction if it is blocked by M. This last action is repeated at most two times, after that the agent turns for the third and last time if it is blocked by M or if it found $o*$ and it will stop as soon as these two cases arise again. An agent stops also every time that it meets another stopped agent. This solution assures to meet all the agents at a node, possibly the special one. Given that an agent turns at most three times the cost of this algorithm is of $O(kn)$ moves where k and n are respectively the number of agents and the size of the graph.
The previous solution cannot be applied to the unoriented ring case because the agents may meet at two different nodes, hence the authors devised a specific protocol to face this problem. The idea of the unoriented protocol is that if the agents are moving in the same direction and M does the same, the agents realize it because the $o*$ node is reached many times. If instead at least one agent moves in a different direction, exactly two groups of agents will be blocked by M at two distinct nodes but, given that k is odd by hypothesis, one of these groups has an odd number of agents, the other one has an even number of agents. This is used in the protocol to break the symmetry, because only the group with an even number of agents moves in the opposite direction to meet the other group. Also for the unoriented case, the complexity of the algorithm is of $O(kn)$ moves.
The second kind of topology analyzed is the mesh. The authors show that in this type of graph there exist non separable starting configurations that do not satisfy the solvability conditions. Among the allowed starting configurations they focus on the ones where the node occupied by the honest agents form a connected component without holes. In the model described above the honest agents have only local visibility, i.e., they only see the other agents at the same node, or M at an adjacent node. This condition is not sufficient to face the rendezvous problem in the meshes, even in the restricted initial configuration studied. To solve the problem an honest agent has to be able to detect the presence of other honest agents at any node within a dis-

tance of two hops, i.e., two edges, with respect to the node where the agent is. However an agent can only count the number of agents at its node and communicate only with these agents. The authors describe an algorithm in which an agent acts according to its current local view. They define eleven possible local configurations of unoccupied nodes, occupied nodes and nodes that can be indistinctly occupied or not. Each agent that executes a view scan is in one of these local configurations and it moves by consequence. The moves ensure that the new disposition of the agents remains a connected component without holes and they gather at a node after $O(k^2)$ moves.

Graph	**Cost (moves)**
oriented ring	$O(kn)$
unoriented ring	impossible for k even $O(kn)$ for k odd
meshes (*)	$O(k^2)$

The size of the graph n and the number of agents k are not known.
* When the initial configuration is a connected component without holes and the agents have a 2-hops visibility.

Table 2.4: Rendezvous complexities in presence of a malicious agent [24].

In Table 2.4 we summarize the results obtained in the article. The protocol for the oriented ring and for the unoriented ring with k odd have a cost of $O(nk)$ moves. The protocol for the meshes has a cost of $O(k^2)$ moves provided that the initial displacement of the agents forms a connected component without holes and the agents have a visibility of two hops.
Studying protocol for meshes in which the agents have a more powerful visibility capability is an interesting open topic. It is also useful to face these problems in the synchronous setting.

Chapter 3

Rendezvous in the Mobile Robots Model

In this chapter we summarize the studies on the mobile robots field mostly related to our work. In the first section, we give a briefly overview on this topic. In the second section, we discuss some works that simplify the mobile robots model to solve the rendezvous in situations unsolvable in the standard model. The third section concerns the more realistic models where the robots occupy a physical space, analyzed from a theoretical point of view. The fourth section is about fat robots protocols studied from a practical point of view, by taking advantage of programming or real robots simulations.

3.1 The Rendezvous Problem in the Robots Model

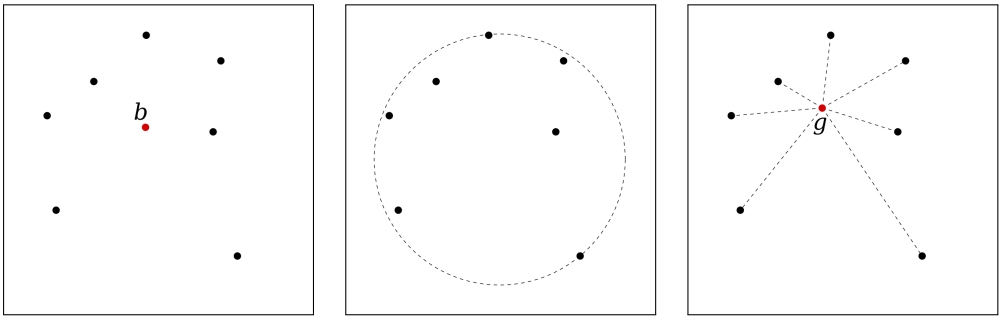
Despite its simple definition, the rendezvous problem is particularly challenging to be solved in the mobile robots model. The objective of the research in the distributed algorithm area is determining exactly under what minimal assumptions on the robots capabilities this problem is solvable.
One of the capability mostly studied is the robot visibility. Many works analyze the rendezvous problem under the hypothesis that the visibility of the robots is limited to a certain constant [5, 12, 33]. This scenario is studied under an obvious assumption.

Definition 3.1.1. *The* ***visibility graph at time t****, G_t, of the robots is the graph having the robots as vertices and an edge (r_i, r_j) if the robot r_i sees r_j*

at time t and vice-versa (typically it is assumed that robots have a view of 360°, thus r_i sees r_j if and only if r_j sees r_i).

In order to solve the rendezvous problem, G_0 must be connected. In other words, at the beginning of the protocol, there cannot be a robot that does not see any other robot because, intuitively, this robot could move away from all the others without ever seeing them.

The first works study the model with the minimal assumptions: anonymous and oblivious robots, considered as points, without a common coordinate system. This scenario is easily solvable in the *FSYNC* setting, without any additional assumptions even if the robots have limited visibility (see e.g., [5]). Conversely the *SSYNC* setting, and by consequence the *ASYNC* one, have been proved to be unsolvable, even if the robots have an unlimited visibility [39].



(a) Barycenter (b) SEC (c) Geometric median

Figure 3.1: The barycenter b, Smallest Enclosure Circle (SEC) and geometric median g of a set of points.

The impossibility results for the rendezvous problem are surprisingly if we consider that an apparently similar problem to the rendezvous, i.e., the *convergence* one, can be solved easily. The convergence problem is a relaxation of the rendezvous, where the robots have to converge towards a point, possibly without never reaching it in finite time.
To solve this problem it is enough to move all the robots towards their center of gravity, or barycenter (see Figure 3.1a).

Definition 3.1.2. *Given a set points $P = \{p_1, p_2, \ldots, p_n\}$, their* ***barycenter*** *is $\sum_{p_i \in P}(p_i)/n$.*

The problem of the barycenter is that it is not invariant with respect to the movements of the robots towards it and thus it cannot be applied to solve

the rendezvous.
The Smallest Enclosure Circle is a related concept (see Figure 3.1b).

Definition 3.1.3. *The **Smallest Enclosure Circle** (SEC), of a set of points is the circle with the minimum radius such that all these points are on the circle or inside of its area.*

The SEC has been used in literature when the robots have limited visibility to solve the convergence problem [6], but it is not suitable to solve the rendezvous under the minimal assumptions because it has the same problem of the barycenter.
It is spontaneous to ask whether it exists a point invariant to the moves of the robots that can be computed as a destination point. The answer is positive, there exists a unique point satisfying this condition, the geometric median, called also Weber (or Fermat or Torricelli) point (see Figure 3.1c).

Definition 3.1.4. *The **geometric median** of a set of points is the point minimizing the sum of the distances between itself and all the other points.*

Unfortunately, it has been proved that this point can not be precisely computed for more than four points [8], but it can only be approximated with numerical methods. It follows that the geometric median cannot be employed to solve the rendezvous problem.

The rendezvous problem in *SSYNC* and *ASYNC* can be solved only by changing the minimal assumptions model, namely adding some assumptions or relaxing some of the existing constraints.
A typical hypothesis added to the model is the *multiplicity detection*: the capability of a robot of detecting if there are more than one robot at the same position (note that when the robots are points they can overlap). This has been used to solve the asynchronous rendezvous for example in [17].
With regard to relaxed model, an assumption could be the agreement of the robots on a common coordinate system (see e.g., [33]). An alternative it is considering that the robot are non-oblivious, i.e., they can memorize the positions of the robots seen and exploit this information for computing a new destination point. With this capability it is possible to solve the rendezvous problem, even in the case of two robots, that is not solvable with only multiplicity detection [16].

The study of robots that are no more points but objects with the size of a disk, called fat robots, represents a more involved scenario. This setting has been proposed only recently [18] with the intent of model a situation more similar to the reality. Two issues make this problem particularly challenging. The visibility of one robot can be partially hidden by another robot, and the space

occupied by one robot can obstruct the movement of another one, in fact the robots cannot overlap as they were points. From the last consideration it follows that the definition of rendezvous must change because the robot cannot meet at a point. All the solutions proposed for the rendezvous with fat robots try to create a compact pattern of robots but there is no an agreement on the rendezvous definition.
This scenario is so difficult that the research is based on empirical tests with programming simulations or real robots [12, 38] but the *ASYNC* problem has been formally proved to be solvable only with three and four robots [18]. The situation changes if we consider the robots to be transparent so that a robot can obstruct only the movement but not the visibility of another one. In [15] an algorithm has been proposed to solve this case for $n > 4$ but only for some non symmetric starting configurations.

A final crucial aspect is that, unlike the mobile agents model in which the complexity is measured in term of moves or of time, in the mobile robots model there is not yet a common method to measure the goodness of an algorithm [33].

3.2 Relaxed Models

In this section we discuss two works that solve the rendezvous in the *ASYNC* setting. The former [33] is related to robots with unlimited visibility that have a common sense of direction. The latter [16] considers that the robots have unlimited visibility and they are non-oblivious.

3.2.1 Common Coordinate System

In Flocchini et al. [33] the authors propose a rendezvous solution for asynchronous robot with limited visibility. In this model a set of oblivious robots, represented as points, is able to detect the presence of other robots up to a given radius V. The model includes the strong constraint of the unlimited visibility and it does not contain any other additional assumptions, such as the multiplicity detection. However, the minimal robots model has been relaxed because robots have a consistent compass, i.e., they agree both on the direction and orientation of the axes, but they can use different units of measure. Without this relaxation, a rendezvous algorithm working in *ASYNC* is impossible. The limited visibility is the major problem of the algorithm, without this constraint and with a common coordinate system the solution

would be trivial: all the robots move towards the rightmost and bottommost robot on the plane.

The solution of the authors is based on the same idea applied for the unlimited visibility, in fact the robots gather at the position of the rightmost and bottommost robot, but the algorithm is not trivial. The solution for the unlimited visibility cannot be simply adapted to the limited visibility scenario, moving one robot towards the rightmost and bottommost robot among the robots in its view. In this way the protocol would be incorrect since the visibility graph may not remain connected and the rendezvous may not be achieved. The algorithm here presented is different and it works as follows. Let r be a robot, r can perform four possible actions according to its current view:

- r remains stopped if there are robots to its left or above its vertical axis.
- r moves down towards the nearest robot, if there are only robots below its vertical axis.
- r moves horizontally towards the vertical axis of the nearest robot, if there are robots only to its right.
- r moves diagonally to the right and down towards a particular point H, if there are robots both below its axis and to its right.

The point H is a point on the vertical axis of the nearest robot r' to the right of r, computed taking into account the distance between r and r' in such a way that r, by moving, does not loose the current robots it sees. All the possible actions of one robot have been devised to assure that the visibility graph remains connected regardless of the moves of the robots. In practice if a robot moves towards another one, this one remains stopped or, if it moves, its movement is limited in such a way the two robots will see again. To be more precise, a robot x can reach another one y and since there is no multiplicity detection at the next look x and y do not realize to be at the same point. However, the authors prove that all the robots that x saw and that are not at the same position of x, are still visible by x and consequently by y, and a symmetric reasoning is valid for y, so that the edges of the visibility graph G are changed but G remains connected.
The authors also prove that the algorithm converges in finite time and all the robots meet at the point of the rightmost and bottommost robot of the starting configuration.

Capabilities	n	Memory
common coordinate system limited visibility	≥ 2	$O(1)$

Table 3.1: Rendezvous solution for n robots in the *ASYNC* model proposed in [33].

The authors show that the agreement on a common coordination system is as powerful as the *FSYNC* assumption because it allows to solve the rendezvous of asynchronous robots with limited visibility (see Table 3.1). Further investigations are required to analyze how the robot capabilities, such as orientation, memory and communication, can be employed to solve a same problem and to check how these ones are related.

3.2.2 Non-oblivious Robots

Cieliebak [16] studies how the non-obliviousness capability can simplify the rendezvous problem in the *ASYNC* setting. The features of the robots are the ones of the minimal model: robots represented as points, without a common coordinate system and with unlimited visibility. There are no other additional assumptions such as multiplicity detection. However, unlike the minimal model, the robots are non-oblivious, i.e., a robot memorizes the positions of the others during the previous Look phase and it moves according to their new positions. In this way robots pass from a memory state to another till they achieved the rendezvous state. This behavior is the same of the mobile agents model, but in this case the robots cannot communicate directly. A robot must communicate indirectly, performing some specific move, to notify the others it accomplished a task. This kind of communication is the thinking behind all the algorithms proposed in the article.

The author solves the asynchronous rendezvous for $n \geq 2$, note that the case $n = 2$ is not solvable even by using multiplicity detection [44].
The first problem studied is the rendezvous of two robots, solved with a specific algorithm described in the following. The robots are fully asynchronous, hence they cannot simply move to their middle point since it could change every time the robots look according to their positions. A robot computes the line l that passes trough the initial positions of the robots and it moves a little step d along l in opposite direction with respect to the other robot. When a robot sees that the other moved, it moves perpendicular with re-

spect to l. The distance d may not be equal for both the robots given that they do not use a common unit of measure. For this reason, when both the robots know that the opposite robot moved perpendicularly, they compute their new horizontal distance h. The knowledge of h is notified with a move along the perpendicular of l and when both the robots know h they start moving towards the middle point of h projected on l.
The rendezvous problem for $n > 2$ exploits the Smallest Enclosure Circle (SEC) of the robots. By considering that at least two robots are on the SEC, the solution has been divided in two cases: exactly two robots are on the SEC and more than two robots are on the SEC.
The former case is solved easily with a variant of the previous proposed algorithm that works as follows. All the robots once awake start moving with little steps until they know that each robot moved. The two robots on the SEC move away from its center along the line l which connects their initial positions, so that the size of the SEC changes but not the line l. The other robots move along lines perpendicular to l. When a robot knows that all the others moved it knows also the positions of the line l and of the lines perpendicular to l along which the robots moved. With this information it starts moving towards the intersection of the median perpendicular line with l, if the number of perpendicular lines is odd. Otherwise, if the number of perpendicular lines is even, a robot moves towards the middle point between the intersections of the two central perpendicular lines with l.
In the last possible starting configuration, where at least three robots are on the SEC, the robots gather at the center C of the SEC. First, a robot computes C and starts moving in circle maintaining the same distance from C. Then, the robot moves towards C when it detects that all the other robots moved. The robots move in such a way that they never swap their positions, thus when a robot looks it always realizes if the others moved.

Capabilities	$\boldsymbol{n}$	**Memory**
non-obliviousness	≥ 2	$O(n)$

Table 3.2: Rendezvous solution for n robots in the *ASYNC* model proposed in [16].

The algorithms proposed in the article show that the the relaxation of the oblivious assumption is more powerful than the multiplicity detection, in fact they solve the rendezvous problem in the *ASYNC* setting for any n (see Table 3.2).
In all these algorithms the amount of memory used is linear on the number

of robots n. An open problem is devising algorithms for non oblivious robots that work with constant memory.

3.3 Fat Robots

In this section we propose the works [15, 18] that analyze how the dimensions of the robots can affect the rendezvous problem. The complexity of the problem increases and, according to the definition of the rendezvous employed in this context, there can be impossible starting configurations. These studies are quite involved and present many open questions.

3.3.1 Fat Robots with Visibility Obstruction

The work of Czyzowicz et al. [18] is the first one concerning the rendezvous problem with fat robots. In their model the robots are oblivious and with unlimited visibility. The robots are disks with equal size, they do not have a common coordinate system but they use the same unit of measure where 1 is the radius of the disk. The robots can touch but cannot overlap and their dimension obstructs not only their movement but also their visibility. The robots in fact are not transparent and when one or more robots are between two other robots they can prevent them to see. The rendezvous is achieved when all the robots touch and form a single group or formally when they form a connected configuration (see Figure 3.2).

Definition 3.3.1. *A set of robots form a* ***connected configuration*** *if for any two points of any two robots it exists a polygonal line l such that all the points of l belong to some robot.*

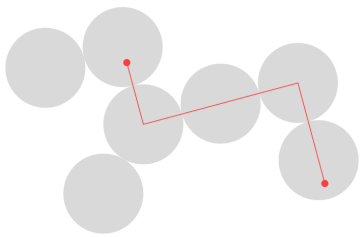


Figure 3.2: An example of *connected configuration* for seven robots.

The authors study the rendezvous problem for three and four fat robots in the *ASYNC* setting provided that each robot knows the total number of robots n. A robot has a *full visibility* if it can see all the other robots in the plane, otherwise it has a *partial visibility*. Note that if a robot is able to see at least two points of another robot border, it can compute the position of this robot, that is the center of its disk. The partial visibility is problematic since it is impossible to agree on a point without knowing the positions of all the other robots. When there are three robots the only possible case of partial visibility is when all the three robots are aligned. In this case the central robot has a full visibility while the others two have a partial visibility. With four robots instead there are three cases: when all the four robots are aligned, when three of the four robots are aligned and when two robots, denoted as central, are between two others, denoted as marginal, in such a way that a marginal robot can not see the other marginal robot. It is evident that the case with four robots is much more difficult than the case with three robots and in fact the relative proposed algorithm in the article is more involved.
The authors first propose a solution for three robots by analyzing the three possible starting configurations and solving each one of them, as describe in the following.
In the first configuration the three robots A, B, C are not aligned and form a triangle where all the angles are smaller or equal to 120°. In this case all the robots can compute the point P such that $\angle APB = \angle BPC = \angle CPA = 120°$ and move towards the point P at distance $2\sqrt{3}/3$. Note that may one robot moves faster than the others but its movement is limited so that the relative angle of the triangle does not increase more than 60°. Therefore the robots starting in this situation continue to form a triangle with all the angles smaller than 120° and continue to compute P. The point P remains invariant with respect to their moves and this ensures they achieve rendezvous regardless of they speed.
When the starting configuration is a triangle with an angle greater than 120°, the robot in this angle remains idle while the others move towards it stopping at distance 2 from it, i.e., moving until they reach it.
The third and last possible case is when the three robots are aligned. In this situation let l be the line that passes trough all the centers of the robots, the central robot, the only one with full visibility, moves tangential to l up to a maximum distance of 1, the other two robots remain idle. When the central robots moves the robots form a triangle where the angle relative to the central robot is the greatest one and, due to the move limitation, it remains greater than 120°. If the central robot moves faster than the external ones, it waits according to the previous configuration. When the external robots move may the central robot is not yet stopped and they continue to correct

their destination points according to the previous configuration until they reach it. This could happen possibly before that the central robot moved for one unit. This fact shows that applying a specific movement could transform one configuration into another one, in this case the third configuration into the second one. This reasoning is applied many times in the protocols for four robots that is split in much more cases.
We briefly describe how the rendezvous for four robots presented in the article works. The idea is that two situations mainly occur: three robots form a triangle with the fourth robot inside it, or four robots form a convex quadrilateral, i.e., a polygon with all the internal angles smaller than 180°. The former case is solved by moving the three external robots towards the central one. The latter is solved by moving the robots along the diagonals of the quadrilateral and in this case the solution is easier if the diagonals are perpendicular.
These two situations are not the only ones, in particular the set of all the possible initial configurations can be split in nine situations, each of which is associated to a specific procedure that robots have to follow. Every robot identifies one configuration according to its current view. The moves depends on the position of the robot, note that, except when the four robots are aligned, two robots have a full visibility and they can move differently from the others. The protocol assumes that rendezvous is achieved with all the robots having full visibility so that they are aware of it. Due to the completely asynchronous movements of the robots, a robot could follow one procedure while another robot is following a different one, but the authors prove that the moves performed by a robot during a procedure never obstruct the moves of another robot. Moreover, the majority of the procedures lead to one or more other procedures before to converge. The transition from a configuration to another one is not deterministic but depends on the speeds of the agents. The authors list and prove all these possible transitions, showing that eventually the robots execute a procedure in which all of them have full visibility and then they terminate in the rendezvous state.

Feature	**Rendezvous Pattern**	$\boldsymbol{n}$
fat robot visibility obstruction	connected configuration	$2 \leq n \leq 4$

Table 3.3: Rendezvous solution for n fat robots in the *ASYNC* model proposed in [18].

The article introduces for the first time the concept of solid robots. The

nature of rendezvous in this context changes so much that it is difficult to generalize the results obtained like in the dimensionless robots model. The authors have devised a specific solution for up to four robots (see Table 3.3). Generalizing these solutions for any finite number of robots, or even devising a specific solution for some $n > 4$ are nowadays unsolved problems.

3.3.2 Transparent Fat Robots

The work of Chaudhuri and Mukhopadhyaya [15] study the rendezvous problem with asynchronous fat robots. The robots are oblivious and with unlimited visibility. They do not agree neither on a common coordinate system nor on a common unit of measure, but they are disks of a same size. These robots are assumed to be transparent. With this assumption the authors get rid of the visibility obstruction problem that characterizes the work about non transparent fat robots of [18]. In this scenario a robot see all the others with a fully visibility regardless of the positions of the robots. However the collisions while the robots move represent a problem also in this model.
Given a set of robots, the authors define the rendezvous as the pattern obtained with a central robot and the other robots placed around in more concentric circles called layers, without holes in a layer, except possibly the last one (see Figure 3.3). This kind of rendezvous is not achievable with two, three or four robots since in these cases it is impossible to distinguish one central robot from the others.

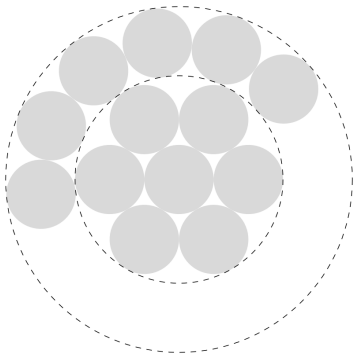


Figure 3.3: An example of the gathering pattern used in [15] for thirteen robots.

The authors propose an algorithm in which one robot at a time moves towards a particular destination creating step by step the rendezvous pattern. This is obtained applying a leader election technique and assuming that some initial configurations cannot be solved. The proposed solution is quite involved, we briefly summarize the key points in the following. The basic idea of the algorithm is that all the robots meet at the center C of the SEC of their starting configuration. The robot elected as leader is the one nearest to C that starts moving towards C and it becomes the central robot of the rendezvous pattern. While this robot is moving it remains the robot nearest to C and thus no other robot is elected in the meanwhile. The election procedure is repeated many times discarding the robots that already reached their final positions, so that one robot at a time moves towards one of the position of the rendezvous pattern. Note that when a robot inside the SEC moves towards its destination the SEC remains invariant. Furthermore, the robot is never obstructed by another moving robot because it was the closest to its destination. A particular case, resolved separately, arises when a robot moves from the border of the SEC since in this case the size of the SEC changes and the next robots that move (at least one) compute their destinations according to the robots already gathered.
The problem of this approach is that when more than one robot is at the same distance from the point C, a unique leader must be electable according to some other feature, such as the the angles of centers of these robots with respect to C. However if two robots have exactly the same view there is no feature that allows to distinguish them and the leader election becomes impossible. This happens when the starting configuration of the robots is straight symmetric or skewed symmetric with respect to a line. In this case the robots detect the impossibility of achieving rendezvous during the first look and fail the goal.

Feature	**Rendezvous Pattern**	$\boldsymbol{n}$
transparent fat robot	concentric pattern	$n \geq 5$

Table 3.4: Rendezvous solution for n fat robots in the *ASYNC* model proposed in [15].

In the article an algorithm to solve the rendezvous problem for any $n \geq 5$ has been proposed (see Table 3.4). This result, compared to the more complex algorithm in [18] to gather only four robots, suggests that the main difficulty to face with the fat robots is the visibility obstruction and not the moving collision. Regarding the collision the authors overcome this aspect by moving

each robot at a time. Even if there exists not yet a common complexity measure for the robots rendezvous algorithms, it is clear that this solution converges slowly because the last robot that moves must wait that all the others moved. An interesting aspect would be devising a faster solution.
Moreover, the proposed solution is based on a election technique that cannot solve the starting symmetric configuration, an open problem is finding a solution for transparent fat robots that works with any initial configuration.

3.4 Robot Simulations

The works of [12, 38] focus on the practice implementation of the protocols for fat robots. Their aim is not to formally prove the correctness of their protocols but to show they are reliable for a real scenario. In [12] the authors simulate an algorithm with *MATLAB*, in [38] the simulations have been made with real *LEGO* robots.

3.4.1 Programming simulation

Before introducing the work of Bolla et al. [12] about fat robots, it is useful to summarize a previous work to which it is connected, namely the rendezvous for robots represented as points of Ando et al. [5].
This work presents a *FSYNC* algorithm where each robot is a point, oblivious and with a visibility of 360° limited to a radius V. The algorithm proposed is the following.
At each instant t a robot r_i moves from its current position $r_i(t)$ towards the center of its SEC denoted as $c_i(t)$. The robot r_i moves over a distance that must be bounded in such a way that the visibility graph at time $t+1$, G_{t+1}, remains connected (provided that G_0 was connected) or, in other words, that all the robots that are mutually visible at time t remain visible at time $t+1$. This distance is computed by considering all the robots visible by r_i. Let r_j be one of these robots, m_j the middle point between $r_i(t)$ and $r_j(t)$ and D_j a circle with radius $V/2$ and center in m_j. To maintain a mutual visibility both r_i and r_j should move inside the circle D_j. This is intuitive because neither r_i nor r_j know where the other robot moves, but if they move at most up to the border of D_j their next distance will be at most V and so they remain visible. Let $\overline{m}$ be the segment joining $r_i(t)$ and $c_i(t)$, the minimum distance that r_i can travel to maintain the visibility with r_j, called l_j, is the length of the segment that starts from $r_i(t)$ and ends in the intersection between $\overline{m}$ and

the circumference of D_j (see Figure 3.4). Let θ_j be the angle $\angle c_i(t)r_i(t)r_j(t)$, l_j is computed as follows:

$$l_j = (d_j/2)\cos\theta_j + \sqrt{(V/2)^2 - ((d_j/2)\sin\theta_j)^2} \tag{3.1}$$

By considering that robot r_i computes l_j for all the visible robots r_j and it cannot move more than the smallest among these distances, r_i moves over distance $min(|\overline{m}|,\ min_{r_j} l_j)$.
The movement described above allows to gather all the robots at a point in a finite time.

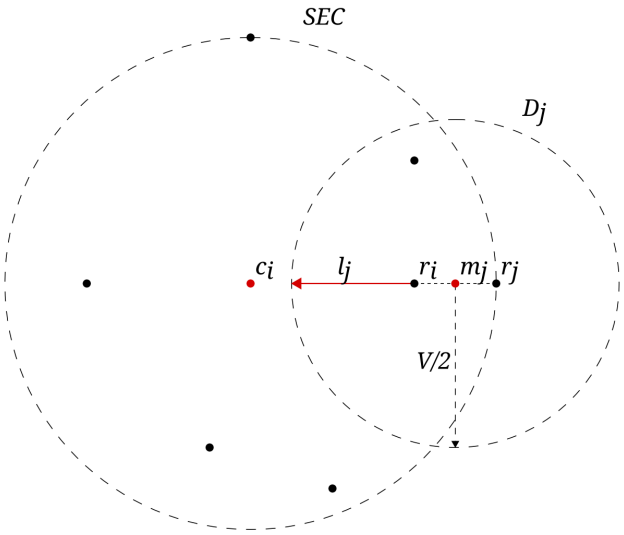


Figure 3.4: A robot r_i and the robots it sees with a view of radius V at time t. If the robot r_i moves as shown with the red arrow, the robots r_i and r_j remain mutually visible at time $t+1$.

The paper of Bolla et al. [12] studies the rendezvous of fat robots in the *FSYNC* setting with a visibility of 360° but limited to a radius of size V. Robots are represented as disks, they are not transparent and so one robot could obstruct the visibility of another one. The robots do not have a common coordinate system and they are oblivious. The collision of the robots is problematic when one robot is blocked by another one and vice-versa, namely when it creates a deadlock situation.
The authors adapt the solution of [5] to the fat robots scenario, note that in both works the robots move synchronously. First, they test directly this

solution without solving the collision problem. Then they apply a slightly modification to this algorithm, introducing a deadlock avoidance technique. Finally, they devise a new protocol that avoids deadlocks and differs from the previous ones for the way in which the robots move.
In the solution proposed in [5] a robot surrounded by some other robots in all the directions move less than a robot located at the margin of a group of robots. This is a consequence of the fact that in the first case the center of the SEC is closer to the robot than in the second case. The novel algorithm enhances this feature. The authors classify the robots in two categories: a robot is called *perimeter robot* if all the robots it can see are inside an angle of 120°, otherwise it is called *inside robot*. Both the inside and the perimeter robots compute the point to move in the same way, but the perimeter ones move over a greater distance. Let r_{di} the furthest robot that a robot r_i can see. In the proposed algorithm the destination point g of r_i is not the center of the SEC of r_i, but it is a point on the segment that joins $r_i(t)$ and $r_{di}(t)$, computed as:

$$g = c\frac{r_{di}(t) - r_i(t)}{V}(r_{di}(t) - r_i(t)) \tag{3.2}$$

Where $c = 1$ in case of a perimeter robot, $c = 1/2$ otherwise.
Suppose for example that r_{di} is on the perimeter of the visibility radius and r_i is a perimeter robot, in this case the destination point of r_i is exactly $r_{di}(t)$. In any case, the movement is bounded in order to preserve the connectivity of the visibility graph. This is obtained as described in the Equation 3.1, with $\theta_j = \angle r_{di}(t)r_i(t)r_j(t)$.
The destination point is modified if a robot detects the presence of another robot in its travel. In this case, the robot changes its destination point with a move called *slip*, avoiding the creation of deadlock situation in which two or more robots mutually block. The new destination point is obtained by the projection of the original destination point on the line tangential to the blocked robot.
The given solution has been empirically tested with a *MATLAB* simulation. The test consisted in scattering on a plane a set with an increasing number of disks, i.e., robots, and applying the algorithms mentioned above: the algorithm of [5], the same algorithm with the addition of the slip procedure and the novel algorithm. Two parameters have been considered to measure the quality of a solution: the time taken to converge and the longest diameter of the pattern of robots obtained. The novel solution is slightly slower than the other two algorithms but it gains a better result with respect to the shape of the group of robots. This is probably due to the fact this algorithm forces to move more the perimeter robots creating a more compact gathering group.

3.4.2 Real Robots Simulation

The work of Pásztor [38] is the most practical among the ones presented in this chapter. The author simulates the rendezvous problem in a synchronous setting with MATLAB and with real LEGO Mindstorm NXT robots. These robots are small devices, quite cheap, with a CPU, a memory, and a set of sensors and motors that can be attached to them in order to implement the robot capabilities (see the next chapter for the hardware and software details). The robots are two-wheel drive and use a third wheel to stabilize. The visibility is given by three ultrasonic sensors that allow to see with a range of 160° and up to 254 cm. To obtain a view of 360° a robot turns over. The author considers the visibility limited to two meters since beyond this distance the ultrasonic sensor is not reliable. The robots are not equipped with a compass, but they have a common notion of the left and right directions. Therefore, they do not agree on the direction of the axis but only on their orientation. The robots should have the same unit of measure given that they use the same motors at the same speed and they detect the other robots by using the same ultrasonic sensors. However the sensors could measure slightly different values, negatively affecting the behavior of the robots. To reach a destination point, a robot must know how much it moved and its current rotation angle with respect to its starting position. To accomplish this task the robot accesses to the information relative to the motors. In particular, the robot samples with a fixed small interval the rotation angle of each wheel. Given the data sampled, the radius of the wheel and the distance between the two wheels, it is possible to compute the distance traveled by the robot and its angle of rotation, that is the new position of the robot while it is moving. This method of navigation, called *odometry*, exploits basic geometric concepts (for an introduction to the topic see [37]). The robots are equipped with a sound sensor to detect the sound over a certain frequency. A sound signal is used to notify the start of the protocol to all the robots, i.e., the time t_0 of the global clock, then every robot relies on an inner counter to remain synchronized with the others. The robots do not apply any types of direct communication, remaining autonomous as provided for in the theoretical model.
The robots are not disks, their shape is rectangular. However, the author considers the rendezvous as a connected configuration, (see Figure 3.2). In this case, the disk is a small circular area around a robot, left empty to be sure the robots do not collide. In fact, a collision is problematic in this context, because the robots could move or overturn jeopardizing the achievement of their task.

Except for the odometry computations the robots can be considered oblivious, in fact they do not store any data obtained in a look phase, following an oblivious protocol.

The protocol used is similar to the one of [5] described previously. The robots compute their destination with the Equation 3.1. Like in [12] the robots follow a collision avoidance technique. When a moving robot detects that another robot is crossing its way, it recomputes its destination point as the projection of the current destination point to its y axis. As in the previous work, the author does not formally prove the correctness of the collision avoidance technique, showing its effectiveness with empirical tests. Before testing the protocol with real robots, the author simulates it with *MATLAB*. In both cases, three starting configurations have been tested: with three, six and ten robots located at the vertices of a regular polygon. The results of the simulations with real robots differ from the others, in particular with ten robots. In fact, one of the robot did not compute correctly its destination point and, by consequence, also the robots near to the faulty robot moved towards different destinations, so that the rendezvous pattern obtained is not circular as expected. The author claims the error is probably due to an inaccuracy of the motors tachometers of the faulty robot.

The robots simulations have been made only with few configurations. These configurations are regular and the rendezvous could be achieved by the robots going straight on, in effect they turned only when they were stopped during the look phases. With more complex configurations the robots should move in all the directions, increasing the complexity of the task. These tests remain an intriguing open problem.
We have to consider that, even with simple configurations, the robots did not follow exactly the theoretical behavior, due to the presence of faults. This fact is interesting because it highlights the necessity of studying more realistic models, taking into account the possible faults occurring in a real scenario.

Chapter 4

Rendezvous despite a Malicious Agent

In this chapter we extend the work of [24] relating to the rendezvous problem despite a malicious agent in an asynchronous setting in rings and meshes. We study the same problem in rings but in a synchronous model, presented in the first section. In the second section, we study the problem under the assumption that the agents agree on a common coordinate system. In the third section, we show that the problem becomes significantly more complex and there exist unsolvable cases. In the last section we summarize the results obtained, by comparing them with the ones of [24].

4.1 Synchronous Model

In this section we present the model we use in all the chapter. This model is the same of [24], except that the agents are synchronous.

We assume that in the network there are k *honest* anonymous identical synchronous agents and one *malicious* agent M which is trying to prevent the gathering of honest agents. More precisely, the network, and the capabilities and behavior of honest and malicious agents are the following:

Network: We consider an undirected ring network $R = (V, E)$, with $n = |V|$ nodes. We say that each edge is connected to a node trough a *port*. The two ports of a node are labeled with a distinct number, 0 or 1, that an agent can distinguish. An agent arriving at a node from the port p moves to the

next node trough the port $(p + 1) \mod 2$, or it moves trough the port p when it reverses its direction. When a protocol begins, we assume that an agent moving clockwise leaves the starting node from the port 0, and an agent moving counter-clockwise leaves it from the port 1. We distinguish two types of rings: *oriented* and *unoriented*.

Definition 4.1.1. *The ring R is **oriented** if the port labeling is globally consistent, i.e., the two ports related to a link are labeled with two distinct values.*
*The ring R is **unoriented** if the port labeling is not globally consistent, i.e., the two ports related to a link can be labeled with the same value.*

In the oriented ring the agents agree on a common coordinate system, i.e., all the agents moving in a direction (clockwise or counter-clockwise) are really moving in the same direction. Conversely, in an *unoriented* ring each agent has a local sense of direction.
We call a node v *occupied* when one or more honest agents are in v, and we call v *free* or *unoccupied* otherwise. The ring contains a specially marked node, $\otimes$, that the agents can distinguish. Without this node the rendezvous problem is not solvable even in the oriented ring case. In fact, suppose that all the agents are moving in a same direction and also M continues to move in this direction. If all the nodes are equal, an agent that has visited all the nodes of the ring cannot realize this fact, thus it continues to turn indefinitely. This problem is solved if there is a marked node, since when an agent reaches it for the second time, it knows it has already visited all the nodes of the ring.
The edges of the network are FIFO links, meaning that all agents that move in the same link respect a FIFO ordering.

Honest agents: The agents are anonymous processes with a limited amount of memory. They start a protocol at a same time, in the same state, and in different positions of the graph, at most one agent at a node. The agents do not know neither the size of the ring n, nor the number of agents k.

They are not equipped with a token and they can communicate only if they are at a same node. The agents are synchronized by a global clock. At each unit of time an agent can read the states of the other agents at its same node, modify its state or the state of the others, decide to move to an adjacent node or to remain stopped. If the agent attempts to visit a node where it is located M it receives a signal that M is at this node at it cannot perform the move, in this case we say that the agent is *blocked* by M.

Malicious agent: The malicious agent M is an omniscient agent with a potentially infinite memory, i.e., it knows n, k, the positions of all the agents, their current states and the protocol they are following. The agent M is asynchronous, however, to make the rendezvous problem solvable, M cannot move at a node occupied by an agent and cannot move on a edge on which an agent is moving. In practice, M can either stay at its current node y or decide to move to another node w, if there is a path from y to w, such that no node on this path (including w) is occupied by any agent. Given that the honest agents moves synchronously, M cannot block simultaneously two agents moving from two different nodes, unless these nodes are the two ones adjacent to the node occupied by M, and in this case we say that M is *surrounded*.

4.2 Oriented Ring

In this section we show that when the agents agree on a common coordinate system the rendezvous problem can be solved in our model quite easily. We provide a novel algorithm that works regardless of the size of the graph and the number of agents and we prove its correctness and its complexity.

The oriented ring case can be seen as a particular case of the unoriented ring one in which all the agents move in the same direction, thus the protocol the unoriented ring that we will propose in the next section can be applied also for the oriented one. However, it is possible to devise a simpler protocol specific for the oriented ring that converges quickly. Also, there is no unsolvable situation in this case, unlike the unoriented ring case as we will analyze in the next section.
To solve the rendezvous in this scenario we propose the Algorithm 1, called *AlternateWalk*, that works as follows. Suppose all the agents start moving in the CW direction, then the first agent that is blocked by M, changes direction, becomes the leader and moves to a state $WALK\_CCW$. Then, it stops the first agent that it meets that is moving in the CW direction (i.e., this agent becomes $STOPPED$), and it continues to move until it is blocked again by M. Finally, it reverses direction and moves back, and the rendezvous is achieved at the node where the leader stopped the first agent that was in the opposite direction. This solution works if two key points are guaranteed: M must be forced to block an agent that becomes the leader; the leader has to meet at a node the agent that is moving in the CW direction. The former point is obtained by blocking one agent at the node $\otimes$ (and letting it become

STAR). If another agent arrives at $\otimes$, the new arrived becomes *STAR* and the other proceeds. This is due to the fact that we want to speed up the meeting between agents moving in a *CW* direction and the one moving in the *CCW* direction. The latter point is obtained by moving the leader agent and the other agents in the *CW* direction at alternate rounds. In particular the leader agent moving *CCW* waits at the even clocks and moves at the odd clocks, vice versa the agents moving *CW* move at the even clocks and wait at the odd clocks. Moreover, after the leader meets the first agent, moves without waiting, thus speeding up the procedure and collecting agents while coming back. We call *WALK* the moves in which we wait and move or vice versa, and *RUN* the moves performed at each clock (i.e., the ones with no waiting time).

Algorithm 1 *AlternateWalk*

```
   #  Algorithm for gathering k ≥ 2 agents in Oriented Ring.
   #  Initial state is WALK_CW, count = 1

 1: count := (count + 1)  mod 2

 2: if State := WALK_CW then
 3:     Dir := CW
 4:     if meet a WALK_CCW agent   then State := STOPPED
 5:     else if meet a RUN_CW agent a then State := FOLLOWER of a
 6:     else if count = 0 then
 7:         if Blocked then
 8:             State := WALK_CCW
 9:         else
10:             Move to the next node
11:             if meet a WALK_CCW or a STOPPED agent   then State := STOPPED
12:             end if
13:             if Reached ⊗  then
14:                 if meet a STAR agent a then
15:                     change the state of a to WALK_CW
16:                 end if
17:                 State := STAR
18:             end if
19:         end if
20:     else
21:         Wait (1)
22:     end if
23: end if

24: if State = STOPPED then  Wait (1)
25: end if
```

AlternateWalk algorithm - Continue

```
26: if State := WALK_CCW then
27:     Dir := CCW
28:     if count = 0 then
29:         Wait (1)
30:         if meet a WALK_CW agent  then
31:             State := RUN_CCW
32:         end if
33:     else
34:         Move to the next node
35:         if meet a WALK_CW or a STAR agent  then
36:             State := RUN_CCW
37:         end if
38:     end if
39: end if

40: if State = RUN_CCW then
41:     Dir := CCW
42:     if Blocked then State := RUN_CW
43:     else
44:         Move to the next node
45:     end if
46: end if

47: if State = RUN_CW then
48:     Dir := CW
49:     if Meet some agents in state STOPPED then
50:         State := RENDEZVOUS
51:         Change the state of the STOPPED agents to RENDEZVOUS
52:     else Move to the next node
53:     end if
54: end if

55: if State = STAR then
56:     if meet a WALK_CCW agent  then State := STOPPED
57:     end if
58:     if meet a RUN_CW agent a then State := FOLLOWER of a
59:     else Wait (1)
60:     end if
61: end if

62: if State = FOLLOWER then
63:     if leader state is RENDEZVOUS then State := RENDEZVOUS
64:     else follow the leader
65:     end if
66: end if
```

Theorem 4.2.1. *The AlternateWalk algorithm achieves the rendezvous of $k \geq 2$ agents in an oriented ring with one malicious agent.*

Proof. Let A be the agent that precedes M, and B the agent that precedes A in the clockwise direction. We prove that A is the unique agent that becomes the leader, i.e. the unique that becomes *WALK_CCW* and, by consequence, *RUN_CCW* and *RUN_CW*.
Let us prove by contradiction that this is not true. Given that all agents are moving in the same direction, links are FIFO, and A is the one that proceeds M, then any other agent may become the leader only if it can pass A. However, A stops only if it reaches $\otimes$, and in this case any other agent that arrives has to stop, it becomes *STAR* and A proceeds, thus not other agent may pass A. Note that only the leader can unblock than agent in $\otimes$ while being in state *RUN_CW*. On the other hand, if any agent stops in $\otimes$, M will be blocked and consequently A will be blocked and will become the leader, thus a contradiction.
Then, A becomes *WALK_CCW*, it reverses its direction and starts moving towards the agent B. Let us first consider the case in which B does not meet $\otimes$. A and B are walking in opposite directions and M is not in the path between them, thus A and B will get closer and closer to each other. When they are at distance one, if it is at an even clock A waits while B moves, and thus the two agents meet at the node of A. Conversely, at an odd clock A moves while B waits, so that the two agents meet at the node of B. If on the other hand B is at $\otimes$ while A is moving in the *CCW* direction, A can meet B alone in state *STAR*, or it can meet it at an odd clock with another agent C. B has just become *WALK_CW* while C has just become *STAR*. In both cases, A meets at least one agent, becomes *RUN_CCW*, and the agents it meets become *STOPPED*. Note that, all the agents moving in the *CW* direction cannot be blocked by M because first they have to meet the *STOPPED* agents, becoming *STOPPED* in turn.
A keeps on moving *CCW* up to when it bumps again into M, which is blocked by the other *STOPPED* agents, and becomes *RUN_CW* collecting all the other agents achieving rendezvous at the *STOPPED* agents. □

Theorem 4.2.2. *AlternateWalk algorithm requires constant memory, converges in $O(n)$ time, and with $O(kn)$ total moves.*

Proof. The worst case on the number of moves is given by the case in which there are only 2 agents and M, and the agents start as far as possible from the node $\otimes$ (i.e., in the next position following the *CW* direction). This is due to the fact that when the leader meets any other agent in state *WALK_CCW*

it moves to state *RUN_CCW* and goes faster. So under this setting M tries to escape, A reaches $\otimes$ after $n-1$ times steps and stops, at the next step B arrives, becomes *STAR* while A, that was *STAR*, is again *WALK_CW*. M is now blocked just before $\otimes$, A reaches it after other $n-2$ time steps, is blocked, it reverses its direction, and meets again B after $n-1$ steps and B moves to the state *STOPPED*. In the worst case A has to move for other $n-2$ time steps to be blocked again by M, and other $n-2$ to go back to B in the *RUN_CW* state, and to rendezvous. Thus globally in the worst case we have $O(n)$ time steps.
The worst case for the number of moves trivially derives from the fact that if all the k agents are as furthest as possible from $\otimes$, in contiguous positions, and M escapes. They all move towards $\otimes$ for at least $n-k$ moves up to when A stops at $\otimes$, thus globally $k(n-k)$ moves. But then all the agents, in turn, reach $\otimes$, thus globally $O(k\,n)$ moves which also includes some other constant number of rounds.
The memory used is constant because the agents use only a bit of memory to distinguish the even and the odd clocks. Thus, the algorithm requires $O(1)$ bits of memory, it converges in $O(n)$ time, and with $O(kn)$ total moves. □

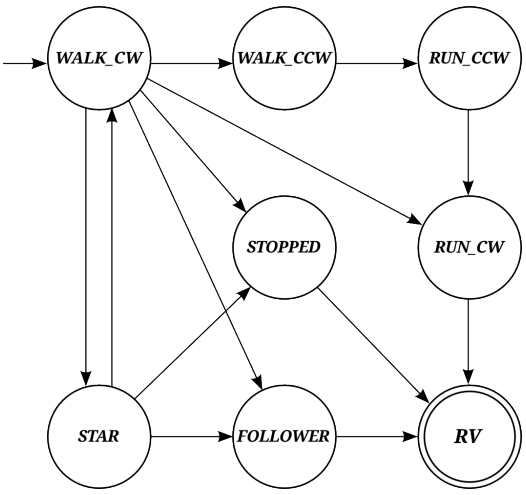


Figure 4.1: Finite automaton of the *AlternateWalk* algorithm, showing the possible transitions from the initial state *WALK_CW* to the final state *RENDEZVOUS* (*RV* in the figure).

Strategy for M. We now show which is the best strategy for the malicious agent M in order to delay as much as possible the gathering of the honest agents. Thus, we will be considering time constraints. Since honest agents stop when they reach $\otimes$, M always tries to escape as much as possible, but

is blocked by an agent stopping in $\otimes$. Then after an agent (the leader) has bumped into M, M tries to escape as much as possible in the opposite direction but it will be blocked again by a *STAR* or a *STOPPED* agent. See Algorithm 2 for more details.

Algorithm 2 Blocking Strategy

```
#  Algorithm for the Malicious Agent in the Oriented Ring
#  Assumptions:  One node is marked ⊗, all honest agents
simultaneously wake up in distinct positions.
```

```
1: if State = INITIAL then
2:     while not blocked by an agent in CW direction do
3:         move to the next node
4:     end while
5:     while not blocked by an agent A in CCW direction do
6:         Wait (1)
7:     end while
8:     while A is moving in CCW direction and not blocked do
9:         follow A in the previous node
10:    end while
11:    State:=STOPPED
12: end if
13: if State = STOPPED then
14:    WAIT (1)
15: end if
```

Lemma 4.2.3. *Algorithm 2 provides the best strategy for the malicious agent to delay as much as possible the gathering of the honest agents.*

Proof. It trivially derives from the fact that the leader has to bump twice into M before collecting the agents, thus M tries to escape as much is possible but it has to stop either when is meets and agent in $\otimes$ or some *STOPPED* agent. □

4.3 Unoriented Ring

In this section we study the rendezvous problem in the synchronous unoriented ring with one malicious agent. In [24] the authors have analyzed the same problem with asynchronous agents and have proved that the problem is solvable iff k is odd. The technique we propose here solves the problem also in the case of n even, k even with $k > 2$, not solved by the algorithm of [24].

Let us first prove some impossibility results. We can use an argument similar to the one used in [24] for asynchronous agents. In addition, in our case, we will use the fact that agents are synchronous. Consider a ring network with an odd number n of nodes, where the port numbers of the ring and initial locations of the agents is chosen by the adversary in such a way that the configuration is symmetric with respect to a line passing through the special node $\otimes$. Each side of the ring is a mirror-image of the other side and there is an equal number of agents in both sides (i.e., k is even). Moreover, the clockwise orientation is in one direction for the agents in one side, and it is the opposite for the other group. The malicious agent M is located on $\otimes$ and blocks any agent from entering it. The agents on one side of the line of symmetry behave symmetrically to their counterpart on the other side. Thus, after each move the configuration will remain symmetric meaning that the only place where agents can gather is on the symmetry axis. However, this is impossible given that the only node on the axis is $\otimes$ which is protected by the malicious agent. Thus for n odd and k even gathering is not possible.

Lemma 4.3.1. *In an unoriented ring with a specially marked node and a malicious agent having arbitrary speed, $k > 2$ mobile synchronous agents starting from arbitrary symmetric locations, cannot gather at a node if n is odd and k is even.*

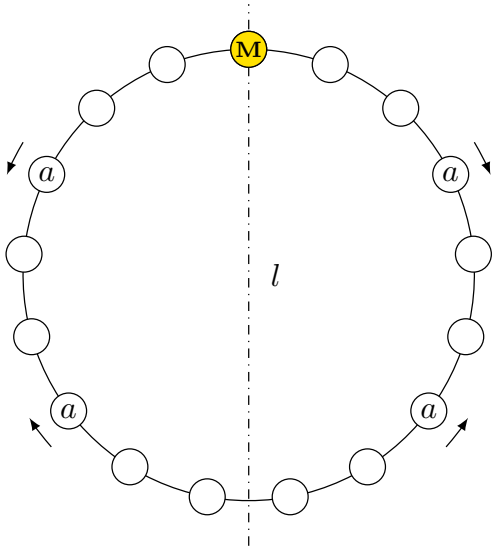


Figure 4.2: An example of unsolvable symmetric starting configuration: the positions of four honest agents denoted by a and their directions are symmetric with respect to the symmetry line l. The malicious agent M is at $\otimes$ (yellow node).

We now propose two algorithms, the first one for $k > 2$ agents, the second one for $k = 2$ agents. Both algorithms assume that the agents are synchronous, n is not known by the agents, the first one also that k is not known. Moreover the ring is unoriented, thus agents do not agree on a common sense of direction, i.e., the clockwise direction might not coincide among all agents. Agents have constant memory. The first algorithm, called *CollectAgents*, works for $k > 2$ agents and it solves the problem for k odd or n even, and for n odd and k even in the configurations in which at most one agent is moving in a direction opposite to the others. When the algorithm cannot solve the rendezvous the agents detect it and exit with a failure. The second algorithm, called *TwinAgents*, solves the problem in the case $k = 2$ and n even.

4.3.1 Unoriented Ring with more than two agents

Algorithm 3, called *CollectAgents*, works as follows. The general idea is that if all agents move in the same direction, either one will stop in *STAR* and will block M that will block other agents, or M protects *STAR* but the agents will be blocked by M and will meet in some other node. The second case is the one in which one group of agents will move in one direction, the other group will move in the other direction and one agent per group will be blocked by M, will stop and will try to surround M. Once M has been surrounded the other agents will collect the surrounding agents, and will gather in a node in the opposite direction. More precisely, an *INITIAL* agent becomes *STAR* if it reaches $\otimes$ twice, or it becomes *STOPPED* if it is blocked by M. When an *INITIAL* agent meets a *STOPPED* agent, it becomes *MSG*, and the other agent becomes *HEAD*. An *HEAD* agent keeps on moving in *CW* direction as much as possible, but it stops if it reaches $\otimes$ twice. Each *INITIAL* agent that meets an *HEAD* agent becomes its *FOLLOWER*. A *MSG* agent reverses its direction, starts moving and while reaching the opposite side it can either be blocked by M, or it can meet a *STOPPED* or a *HEAD* agent. In all cases, the *MSG* agent becomes *R_MSG* and goes back towards the *HEAD* agent, delivering the information it has collected if it has met the *HEAD* agent. A *STAR* agent that meets an *R_MSG* agent becomes its *FOLLOWER*.

If there is only one *HEAD* agent, then rendezvous is achieved when the *R_MSG* agent meets the *HEAD* agent. Otherwise, the two *HEAD* agents act according to the information delivered by the *R_MSG* agents. If the number of agents is odd, the *HEAD* agent of the group with odd agents remain stopped in state *WAIT*, while the other one, with its followers, moves

towards it. Conversely, if the number of agents is even (this is computed using a parity bit), then the *MSG* agents continue to move back and forth in order to notify to their respective *HEAD* agent if the other one has moved or not. When both *HEAD* agents stop, since they surrounded M, then they move in opposite directions with their followers, meeting at a node if n is even or failing if n is odd.

Algorithm 3 *CollectAgents*

```
#  Gathering k > 2 agents in an Unoriented Ring.
#  Assumptions:  All honest agents simultaneously wake up, n
#  and k are not known, one node is marked ⊗, if n is odd and
#  k is even and more than one agent moves differently from
#  the others, the agents do not gather.
#  Local variables:  sync = parity = star_reached = 0,
#  moved = -1, first_clock = 1

 1: sync := (sync + 1) mod 2

 2: if State = INITIAL then
 3:     dir := CW
 4:     if first_clock = 1 and Reached ⊗ then star_reached = star_reached +1
 5:     end if
 6:     if Blocked then State := STOPPED
 7:     else
 8:         Move to the next node
 9:         if Reached ⊗ then star_reached = star_reached +1
10:         end if
11:         if meet a STOPPED agent  then State := MSG
12:         else if meet a HEAD agent h then State := FOLLOWER of h
13:         else if Blocked then State := STOPPED
14:         else if star_reached = 2 and not found a STAR agent then
15:             State := STAR
16:         end if
17:     end if

18: else if State = STOPPED then
19:     if meet a INITIAL agent  then
20:         State := HEAD
21:         moved:= 0
22:         parity:= (parity + 1) mod 2
23:     else if meet a MSG agent m then State := FOLLOWER of m
24:     else Wait (1)
25:     end if
```

CollectAgents algorithm - Continue

```
26: else if State = HEAD then
27:     if meet a INITIAL agent   then parity:= (parity + 1) mod 2
28:     else if meet a R_MSG agent r then
29:         if r.moved = -1 then State := RENDEZVOUS
30:         else if parity = 0 and r.parity = 1 then  State := WAIT
31:         else if parity = 1 and r.parity = 0 then State := R_HEAD
32:         else if moved = 0 and r.moved = 0 then
33:             State := R_HEAD
34:             Wait (sync)
35:         end if
36:         moved:= 0
37:     else if not blocked and star_reached < 2 then
38:         moved:= 1
39:         Move to the next node
40:         if Reached ⊗ then star_reached = star_reached +1
41:         end if
42:     else Wait (1)
43:     end if

44: else if State = STAR then
45:     if meet a R_MSG agent r then State := FOLLOWER of r
46:     else Wait (1)
47:     end if

48: else if State = FOLLOWER then
49:     if leader state is RENDEZVOUS then State := RENDEZVOUS
50:     else follow the leader
51:     end if

52: else if State = MSG then
53:     dir:= CCW
54:     if Blocked then
55:         State := R_MSG
56:     else
57:         Move to the next node
58:         if meet a HEAD agent h then
59:             state := R_MSG
60:             moved:= h.moved
61:             parity:= h.parity
62:         else if Blocked or meet a STOPPED agent   then state := R_MSG
63:         end if
64:     end if
```

CollectAgents algorithm - Continue

```
65: else if State = R_MSG then
66:     dir:= CW
67:     Move to the next node
68:     if meet a HEAD agent h then
69:         if moved = -1 then State := RENDEZVOUS
70:         else if (parity!= h.parity) or (moved = 0 and h.moved = 0) then
71:             State := FOLLOWER
72:         else State := MSG
73:         end if
74:     end if

75: else if State = R_HEAD then
76:     dir:= CCW
77:     if Blocked then State := FAILURE
78:     else if meet a R_HEAD agent  then State := RENDEZVOUS
79:     else
80:         Move to the next node
81:         if meet a R_HEAD or WAIT agent  then State := RENDEZVOUS
82:         end if
83:     end if

84: else if State = WAIT then
85:     if meet a R_HEAD agent  then
86:         State := RENDEZVOUS
87:         Change state of FOLLOWER agents to RENDEZVOUS
88:     else Wait (1)
89:     end if
90: end if

91: first_clock:= 0
```

Theorem 4.3.2. *The CollectAgents algorithm solves the rendezvous problem for $k > 2$ agents in an unoriented ring of size n with a special node $\otimes$, and with one malicious agent M. It returns a failure message when n is odd, k is even and at least two agents move in opposite directions.*

Proof. All the agents start moving at the same time in their *CW* direction, but there is no common sense of direction thus some agents could move in one direction while some others are moving in the opposite one. By considering the direction of one of the groups of agents, three possible cases could arise: *1*) all the agents move in the same direction; *2*) all the agents except one move in the same direction; *3*) at least two agents move in one direction and at least two move in the opposite one.

1) We want to prove that exactly one agent becomes *STOPPED*. Let us

first assume by contradiction that in this case no agent becomes *STOPPED*. There are two cases: a) Either M stops or moves in a direction opposite to the other agents, but in this case it blocks one agent that becomes *STOPPED*, thus a contradiction; b) or M keeps on moving in the same direction chosen by the agents, but in this case one agent will cross $\otimes$ twice and will stop, thus blocking M. Therefore, at least another agent will eventually be blocked by M (no other agent stops at $\otimes$) and will become *STOPPED*, thus a contradiction also in this case. Let us now prove that the *STOPPED* agent is unique. Given that by hypothesis there are $k > 2$ agents that move in the same direction, and at most one agent becomes *STAR*, then one agent m has to meet h, the *STOPPED* one, becomes *MSG*, while h changes its state to *HEAD*. Note that, all the other agents in state *INITIAL* cannot become *STOPPED*, but become *FOLLOWER*, given that they meet h before being blocked by M. The agents h and m move in opposite directions. m reaches M in the opposite side because there are no other *STOPPED* or *HEAD* agents, given that m is the first agent that moves in this direction. Thus, the *STOPPED* agent is unique. Let us now prove that the unique *STOPPED* agent, that is m, will coordinate the gathering. When m comes back in state *R_MSG* has the default *moved* value equal to -1. Agent h in the opposite side continues to move in the same direction, with some agents in state *FOLLOWER*. However, the two agents m and h eventually meet, in fact either h is stopped by M, or after it has reached $\otimes$ for the second time, it will remains stopped, together with the agents in state *FOLLOWER*, waiting for the arrival of m to rendezvous. Note that, if one agent had previously stopped in $\otimes$, then it has to be in the path followed by *R_MSG*, thus it will become a *FOLLOWER* of *R_MSG* and will rendezvous with the other agents. Thus, if all agents initially move in the same direction rendezvous is achieved.

2) Let us assume that all the agents except one, called s, move in the same direction. The agent M cannot escape from s and from the agents going in the opposite direction, thus M has to block both s and another agent h. s becomes *STOPPED* but not *HEAD* because no other agent can reach it in state *INITIAL*, whereas h, becomes *STOPPED*, and once it is reached by an *INITIAL* agent, m, it becomes *HEAD*, and m starts moving in the opposite direction, i.e. towards s, in state *MSG*. All the remaining agents become *FOLLOWER* of h. When s and m meet, s becomes a *FOLLOWER* of m, and m moves in state *R_MSG* towards h with the default *moved* value equal to -1.
The agent h and its *FOLLOWER* continue to move in *CW* direction, however h is either blocked by M or it stops after reaching $\otimes$ twice. When m and s reach h, all the other possible agents already reached h, and since *moved*

$= -1$ they rendezvous.

3) At least two agents move on both sides. Thus, exactly 2 agents become *HEAD* and 2 agents become *MSG*. Let us call them h_1 and m_1 on one side and h_2 and m_2 on the other side, respectively. All the other agent become *FOLLOWER* of the *HEAD* they meet. m_1 and m_2 perform a tour in opposite directions: they reverses direction, reach h_2 and h_1, respectively, and come back to h_1 and h_2 in state *R_MSG* delivering the *moved* and the *parity* information related to the other *HEAD* agent.
We now have to show that the two groups of agents are able to meet. First observe that since agents are synchronous and links are FIFO m_2 (m_1) arrives at h_1 (h_2) in state *MSG* before that m_1 (m_2) comes back in state *R_MSG* delivering the information on the parity or oddness (computed using the *parity* bit) of the opposite group of agents and the information on the eventual movement of the opposite head, stored in a variable *moved*. Note that, when M is surrounded, the heads are not able to move. We have two cases:

- ***k* is odd:** If k is odd, the *HEAD* agent of a group with odd agents, i.e., with *parity* $= 0$, remains stopped in state *WAIT*, whereas the other *HEAD* agent, with *parity* $= 1$, starts moving in opposite direction in state *R_HEAD* with all the other agents of its group in state *FOLLOWER*. This group of agents will reach the other stopped group -regardless of the value n- and all agents will rendezvous.

- ***k* is even:** This case is more complicated and rendezvous can be achieved only when M has been surrounded, i.e., when h_1 and h_2 do not move anymore. This is possible since even if M, is very fast, due to the synchronicity of the agents, can block only h_1 or h_2 in one time step. Note also that each *HEAD* agent cannot meet $\otimes$ twice since M is surrounded before each of them visits all the nodes of the ring.

  Thus, we have now to prove that m_1 and m_2 perform a same number of tours and eventually meet, together with their *FOLLOWER* agents. Given that m_1 and m_2 collect both the *moved* value of their *HEAD* and the one of the opposite *HEAD* agent, if they are both not 0 (which means both *HEAD* agents did not move) they start a new round, otherwise they try to rendezvous. Observe that collecting both values and having FIFO links implies a synchronization of the rounds.
  Let G_1 and G_2 be the two groups of agents surrounding M that will eventually be formed at the node of h_1 and of h_2 respectively, each one with a size of at least 2 agents. The *sync* variable changes at each clock, thus it is used to synchronize the movements. In the case n even h_1 and h_2 are separated by an odd number of nodes. If one *HEAD* agents

changes direction at an even clock and the other at an odd clock (defined by the *sync* variable), only one of the groups waits one cycle of clock, and so the two groups move at an instant of time that preserves the odd distance and thus rendezvous.
Conversely, if n is odd h_1 and h_2 do not meet along the path are are blocked again by M, thus they move to the state *FAILURE* since they did not rendezvous.

□

Theorem 4.3.3. *The CollectAgents algorithm in an unoriented ring of size n requires a constant amount memory and it converges in $O(n)$ time steps and with $O(kn)$ total moves.*

Proof. The amount of memory used by the agents is constant since it does not depend on n or k, but it trivially depends on a constant number of variables which can only take constant values.

The worst case time complexity has to be computed on the three possible settings described in the proof of Theorem 4.3.2. In the first two cases, i.e., when all the agents are moving in the same direction, or when only one moves in the opposite direction, we have that after auto most $n-1$ steps each agent arrives either at *STAR* or is blocked by M. In the first case M might keep on moving around but after at most other n steps at least on agent stops in *STAR*. In all the other cases one agent stops is a shorter time, blocked by M. Then, one agent moves back and forth to collect agents in at most $2 \times (n-1)$ time steps, and then rendezvous. Globally, in these two cases $O(n)$ time steps are required in the worst case.
The third case, the one in which at least two pair of agents move in opposite directions, is more complicated. The first round has the same complexity of the second case, i.e., $O(n)$ time steps. Then, some rounds are repeated, each of which assumes that a message goes back and forth from one *HEAD* to the opposite one. The worst case is the one in which M block the agents as much as possible before it is surrounded. Assume the extreme case in which all agents start in neighbouring positions and recall that at each round M might only block the agent on one side. Thus, when the distance among the agent is x, while the *MSG* are moving, the two heads conquer x positions, thus a bound is on the number of times that a *MSG* agent turns, is given by $\sum_{i=0}^{\log(n)} 2^i = 2n-1$. Then some few extra rounds are required to collect the final information or to fail. Globally the time is $O(n)$ time steps.
A trivial bound on the number of moves is given by $O(k \times n)$ if each agent has execute all steps (i.e., when agents are grouped in block and have to

move together towards M starting from the furthest positions. □

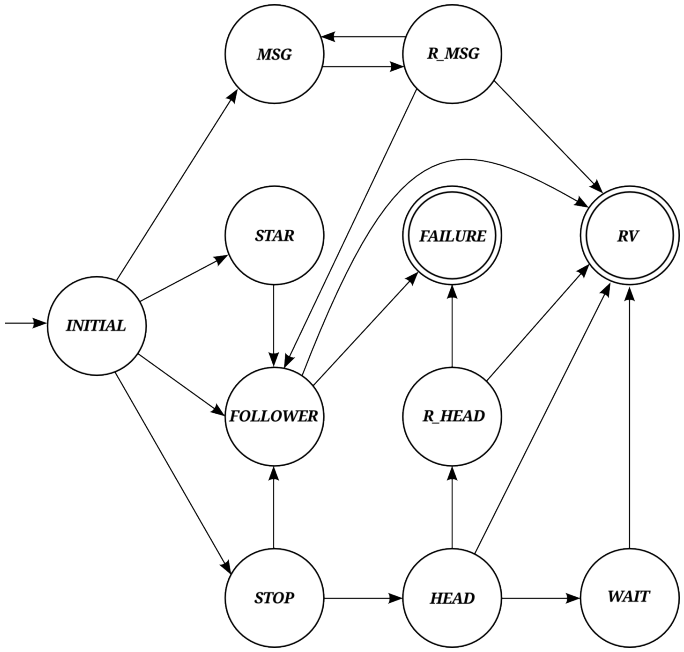


Figure 4.3: Finite automaton of the *CollectAgents* algorithm, showing the possible transitions from the state *INITIAL* to the final states *RENDEZVOUS* (*RV* in the figure) or *FAILURE*.

Strategy for M. We now show which is the best strategy for the malicious agent M in order to delay as much as possible the gathering of the honest agents. Thus, we will be considering time constraints. The idea is that if M detects that k agents have the same orientation, moves in that direction up to when it is blocked by an agent, then it waits that agent moves (once it becomes a *FOLLOWER* of an *R_MSG*) and it follows it, until it is blocked again. This happens after a *HEAD* has crossed twice the node $\otimes$ and it has stopped. If $k-1$ agents have the same CW orientation, it moves CW up to when it is blocked by an agent. Finally, if k is odd, it moves in the direction of the group of even size, if k is even it alternatively blocks *HEAD* agents from one or the other side. See Algorithm 4 for more details.

Algorithm 4 Blocking Strategy

```
#  Algorithm for the Malicious Agent in Unoriented Ring
#  Assumptions:  One node is marked ⊗, all honest agents
simultaneously wake up
```

1: **if** k agents have initially the same CW orientation **then**
2: **while** not blocked **do**
3: move in CW direction
4: **end while**
5: **while** blocked **do**
6: Wait (1)
7: **end while**
8: **while** not blocked **do**
9: move in CCW direction
10: **end while**
11: State:=*STOPPED*
12: **else if** $k-1$ agents have initially the same CW orientation **then**
13: **while** not blocked **do**
14: move in CW direction
15: **end while**
16: State:=*STOPPED*
17: **else if** k is odd **then**
18: **while** not blocked **do**
19: move in the direction of the group of even size
20: **end while**
21: **else if** k is even **then**
22: **if** not Blocked in both directions **then**
23: Blocks in one move a *HEAD* agent in one direction and, in the next move, the other
24: **end if**
25: **end if**

Lemma 4.3.4. *Algorithm 4 provides the best strategy for the malicious agent to delay as much as possible the gathering of the honest agents.*

Proof. The correctness of the algorithm derives from the fact that if all the agents move in the same CW directions for two rounds do not take any action unless they are blocked by M, so M tries to delay the blocking as much as possible. Once M is blocked it waits and it tries to delay the blocking of the leader moving in CCW as much as possible. If only one agent moves in the opposite direction the best strategy for M is to try to escape from the $k-1$ agents since only one of this group will perform the gathering after moving CCW and CW. Thus, this step has to be delayed as much as possible. If the agents move in opposite directions, and k is odd, M tries to delay as much as possible the agent that collects the group of even size. The global time will be given by the time the second *INITIAL* agent meets a *STOPPED* agent

that has been blocked by M (so this has to be maximized), the time the *INITIAL* agent that now becomes *MSG* spends to reach the group of even size ($n-1$ time steps), and other $n-1$ steps with this group to go back. The last case is k even. In this case to maximize the gathering time M has to delay the meeting phase in the middle position, thus at each time step it blocks at least one of the *HEAD* agents. □

4.3.2 Unoriented ring with two agents

We now show how to solve the rendezvous problem in an unoriented ring with $k = 2$ agents, n even. First it assumes that the agents know $k = 2$, but do not know n (although they might use $\log n$ bit in the worst case). The Algorithm 5, called *TwinAgents*, works as follows. It uses the general idea of the Controlled Distance algorithm in a ring [41] of moving back and forth for a certain distance. However, there are many other differences. First, each round is run by both agents A and B, and not by a subset of them. Both agents try to move for x steps in the CW direction, i.e., for x cycles of clock either they move (if possible, i.e., if they are not blocked by M), or they wait. Then, they move back in the CCW direction for x steps or they wait, unless they gather. Note that, if they have waited some time moving CW, then they might go back passing the starting position. At each round i the value of x increases, and we assume $x = 2^i$, for $i = 0, 1, 2, \ldots$. The process stops when the agents gather.

We will prove that eventually the two agents will either meet at some intermediate step, or if they move in opposite directions, they will eventually surround M and, since they start moving in CCW direction at the same time step (i.e., they are synchronized on the reverse move), they will meet at the middle node (opposite to M in the ring). On the other hand, if the agents move in the same direction, they are either blocked by M, and with x big enough will either stop in the same node, or if M keeps on moving in the same direction, to prevent them to infinitely increase the value x, an agent stops when it reaches $\otimes$ twice. In this way the other agent will meet it at the node $\otimes$.

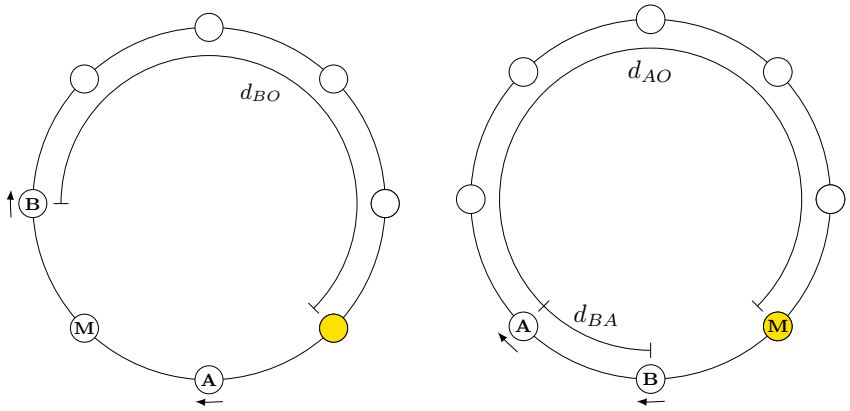
Algorithm 5 *TwinAgents*

```
#  Algorithm for gathering 2 agents in Unoriented Ring.
#  n is even, x = 1, initial state is MOVECW

 1: if State = MOVECW then
 2:     x := 2x
 3:     dir := CW
 4:     count := 0
 5:     moved := 1
 6:     for i := 1 to x do
 7:         if meet a honest agent   then State := RENDEZVOUS
 8:         else if Reached ⊗  and moved = 1 then
 9:             if count = 0 then
10:                 count := 1
11:                 moved := 0
12:             else State := IDLE
13:             end if
14:         else if Not blocked by M then
15:             Move to the next node
16:             moved := 1
17:             if meet a honest agent   then State := RENDEZVOUS
18:             end if
19:         else Wait (1)
20:         end if
21:     end for
22:     State := MOVECCW

23: else if State = MOVECCW then
24:     dir := CCW
25:     for i := 1 to x do
26:         if meet a honest agent   then State := RENDEZVOUS
27:         else if Not blocked by M then
28:             Move to the next node
29:             if meet a honest agent   then State := RENDEZVOUS
30:             end if
31:         else Wait (1)
32:         end if
33:     end for
34:     State := MOVECW

35: else if State = IDLE then
36:     if meet a honest agent   then State := RENDEZVOUS
37:     else Wait (1)
38:     end if
39: end if
```

Theorem 4.3.5. *TwinAgents algorithm solves the rendezvous problem of $k = 2$ agents in an unoriented ring of size n with a special node $\otimes$, and with one malicious agent M.*

Proof. There are two possible situations: *1)* the two agents move in the same direction; *2)* the two agents move in opposite directions.

1) First note that M can block only one agent, A, in the CW direction and the other agent, B, in the CCW direction. Let d_{AM} and d_{BA} be the two distances between A and M and between B and A, measured as the number of edges that separate the two agents in the clockwise direction.



(a) Node $\otimes$ is closer to B (b) Node $\otimes$ is closer to A

Figure 4.4: Notation used when both the agents move in clockwise direction.

When M blocks an agent the distance d_{BA} decreases and the two agents could meet in state $MOVECW$ or $MOVECCW$ if M is blocking A or B respectively. However, M can continue to move in the CW direction to maintain constant the distance d_{BA}.

In this case we show that eventually x is great enough to allow one agent to reach the node $\otimes$ twice during a $MOVECW$ phase. This agent becomes $IDLE$ and we show that the other agent will meet it at $\otimes$. An agent can reach $\otimes$ twice if x is at least equal to n. This happens if the agent starts exactly from $\otimes$ and it is not blocked by M while it moves for n steps in the CW direction. Note that if an agent is at a certain distance from $\otimes$ at a $MOVECW$ phase, the distance may increase at the successive phase because the agent reached $\otimes$ only once. However, in the following we prove an upper

bound on the value of x.
When a *MOVECW* phase starts, $\otimes$ can be closer to B or to A. If it is closer to B, B could reach $\otimes$ twice in CW direction, if it moves at most $d_1 = d_{BO} + n$, where d_{BO} is the clockwise distance from B to $\otimes$. The maximum distance d_1 is $2n - 3$ when $d_{BO} = n - 3$, (see Figure 4.4a).
When $\otimes$ is closer to A, A could reach $\otimes$ twice in CW direction, if it moves at most $d_2 = d_{AO} + n + (d_{BA} - 1)$, where d_{AO} is the clockwise distance from A to $\otimes$ and $(d_{BA} - 1)$ is the maximum number of times that M can block A avoiding that B meets it. Note that when d_{AO} increases, d_{BA} decreases and the maximum d_2 is $2n - 2$ when $d_{AO} = n - 2$ and $d_{BA} = 0$, as shown in Figure 4.4b.
Thus, we can consider $x \geq 2n$ as an upper bound on the number of moves such that one agent reaches $\otimes$ twice, becoming *IDLE*. When the *IDLE* agent is B, M prevents A to reach it moving in the CW direction, but A will meet B once it moves in the CCW direction. When the *IDLE* agent is A, M cannot prevent B to reach A in the CW direction.

2) We prove an upper bound on the value x allowing the two agents to surround M after a *MOVECW* phase, regardless of their starting positions and the behavior of M. Suppose without loss of generality that when the agent A moves in the CW direction, M escapes from A moving in its same direction. However, by considering that the maximum distance between A and M measured in the CW direction is $n - 1$ edges, and the maximum distance such that A and B does not meet in the CW direction is $n - 3$ edges, the number of moves that M can performed is limited, since M moves towards B. These moves are $n/2 - 2$, thus with $n + n/2 - 4$ moves both the agents surround M. Therefore, the upper bound of the previous case $x = 2n$ is valid also for this case and it guarantees that the rendezvous is achieved. In fact, with $x \geq 2n$ the two agents first surround M, then they meet at a node moving in the CCW direction (recall that n is even by hypothesis, so that when they start moving in the *MOVECCW* phase they are separated by $n - 3$ nodes). □

Theorem 4.3.6. *TwinAgents algorithm for $k = 2$ agents on an unoriented ring of size n requires $O(n)$ memory and it converges in $O(nlog(n))$ time and with $O(nlog(n))$ total moves.*

Proof. A linear bound of the memory used, $2n$, comes directly from the previous theorem. Since the agents double x at each *MOVECW* phase, they required $O(log(n))$ phases to obtain $x = 2n$ and a trivial bound of the total time required is $O(nlog(n))$. The upper bound of x is reached when the two agents move in the same direction. Since they move at each cycle of clock

the bound of the time is also the bound of the total number of moves, i.e., $O(nlog(n))$.

□

4.4 Results

In Table 4.1 we summarize the results obtained in our synchronous model and we compare them with the ones of the asynchronous case presented in [24].

Table 4.1: Rendezvous of k agents in a ring of size n.

	Memory	**Time**	**Moves**
Oriented Ring	$O(1)$	$O(n)$	$O(kn)$
Unoriented Ring $k = 2$ and n even	$O(\log(n))$ bits	$O(n \log(n))$	$O(n \log(n))$
Unoriented Ring $k > 2$. At most one agent in opposite direction if n is odd	$O(1)$	$O(n)$	$O(kn)$

In the asynchronous setting, the rendezvous problem is solvable in an oriented ring with constant memory, linear time and with $O(kn)$ total moves, and the agents change directions up to three times before achieving the rendezvous [24]. In the synchronous case one leader agent changes direction at most twice and all the others do not change directions.
In [24] the authors prove that the rendezvous problem in an unoriented ring is solvable if and only if k is odd. In the synchronous case we proved that the rendezvous is unsolvable only for symmetric configurations. In particular, the proposed algorithms solve the problem also for k even, provided that n is even or, if n is odd, $k > 2$ and no more than one agent moves in opposite direction with respect to all the others. The algorithm for $k > 2$ agents uses a constant amount of memory, it converges in time $O(n)$ and with $O(kn)$ total moves, where n is the size of the ring.
Conversely, the algorithm for $k = 2$ (unsolvable in the asynchronous case) uses $O(\log(n))$ bits of memory, and it converges in $O(n \log(n))$ total moves and time. This last result is due to the fact that two agents going in opposite direction do not realize exactly when they surround M, thus they make more

attempts to surround it. The existence of an algorithm that solves this problem for $k = 2$ in constant memory, or in linear time and number of moves remains an open question.

Chapter 5

Real Robots Protocol

In this chapter we discuss how real robots can be used to execute the protocols proposed in the previous chapter. In the first section, we describe our robots and the programming environment. In the second section, we show how the devices of the robots are utilized to implement their capabilities. Finally, the third section concerns the results we obtained and the limits of our work.

5.1 The Hardware of the Robots

We used the LEGO Mindstorms EV3 robots [2]. These robots, commercialized by LEGO since 2013, are the successors of the NXT robots, produced by the same company. An EV3 robot consists of a mini computer, called *Brick*, with these specifications:

- *CPU*: ARM926EJ-S core, 300 MHz
- *Memory*: RAM 64 MB, Flash 16 MB
- *Storage*: Micro SD card up to 32 GB
- *Operative System*: a Linux distribution developed by LEGO

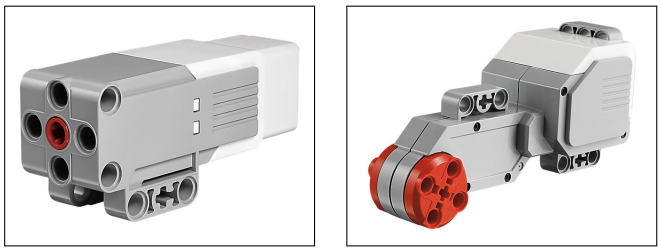
An EV3 robot has eights ports to connect up to four motors and up to four sensors. Motors and sensors are connected with RJ12 cables, and can be assembled to the Brick with the LEGO pieces. EV3 robots support Wi-Fi and Bluetooth connections.

We used these robots because they are quite cheap and widely used in the academy context [35], thus our tests can be easily replicated also by other

authors. Furthermore, these robots allow to program at an high level the plugged motors and sensors, without dealing with their electronic implementation. The drawback is that the robots sensors are not fully reliable, as we will explain later, and this makes particularly challenging the execution of a distributed protocol.
We now describe the hardware of the robots, then we will discuss which of them is suitable for our goal.

Motors
Large Motor: It is a DC motor equipped with a tachometer. It can be set to run for a certain number of cycles, for a precise degree of rotation with an accuracy of one degree, or for a given amount of time. It is possible to set the number of revolutions per minute (RPM), that determines the speed of the motor, up to 170 RPM, as well as the polarity of the motor (clockwise or counter-clockwise rotation). It has a running torque of 20 N/cm and a stall torque of 40 N/cm.

Medium Motor: It is a motor less powerful than the Large Motor (running torque of 8 N/cm and stall torque of 12 N/cm) but it can reach an higher speed, up to 250 RPM. In general the Large Motor is used if the movement is subjected to high friction, like the move of a wheel that runs on a surface, while Medium Motor is used for simpler tasks, like the move of the robot head.



(a) (b)

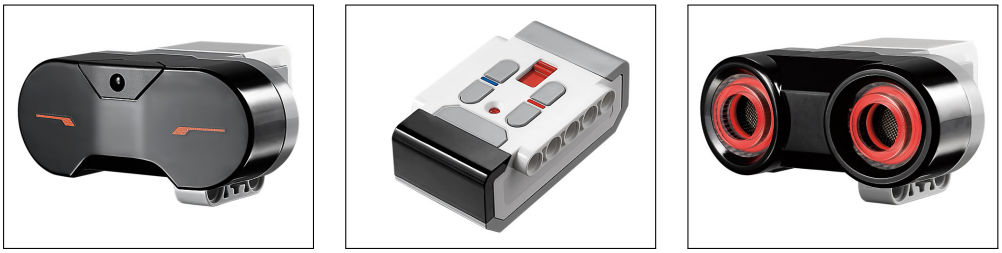
Figure 5.1: LEGO Mindstorms EV3 Large Motor (a) and Medium Motor (b) (Source: shop.lego.com).

Sensors
Infrared Sensor: this sensor measures the distance of an object up to approximately 70 cm. The sensor consists of a LED and a light sensor. The LED produces a light with a wavelength in the infrared spectrum. This light bounces off a close object into the sensor that, according to the light intensity, determines the distance of the hit object. This sensor can also detect

the light emitted by a remote control, the Infrared Beacon, as described in the following.

Infrared Beacon: this sensor is a remote control that must be used with the Infrared Sensor. It can emit an infrared light in four different channels, i.e., frequencies, that is detected by the Infrared Sensor up to two meters of distance. The beacon needs not to be exactly in front of the Infrared Sensor, but it can be located sideways with respect to this one. The Infrared Sensor detects the distance and the orientation of the beacon. The channel can be set only manually with a specific button. The Infrared Beacon contains four other buttons, and for each channel the Infrared Sensor can detect if one button has been pressed, or two buttons have been pressed simultaneously. Alternatively, by pressing an apposite button, the beacon transmits the light continuously.

Ultrasonic Sensor: this sensor measures the distance of an object between 1 and 250 cm with an accuracy of +/- 1 cm. The sensor can act as an active or passive sonar. In the former case the sensor emits sound waves and listens their echoes to compute the distance from objects. The measurement can be continuous if the sensor continues to generate sound waves, or it can be single if the sensor takes a measure and then it turns off. The sensor applied in passive mode, listens a sound emitted from another source, that is the sound generated by another ultrasonic sensor or a loud noise such as a clapping. In this case the sensor returns true if a sound has been detected, false otherwise.



(a) (b) (c)

Figure 5.2: LEGO Mindstorms EV3 Infrared Sensor (a), Infrared Beacon (b) and Ultrasonic Sensor (c) (Source: shop.lego.com).

Color Sensor: this sensor measures the color of a close surface. The sensor consists of three LEDs, red, green and blue, that it uses to illuminate an area, retrieving its color with a sample rate of 1kHz. By changing the way in which the sensor projects the LED lights, the color information sampled changes. In particular, the single red LED is used to determine the amount

of light reflected. The same task is performed with the blue LED in case of dimly lit. In these cases the sensor distinguishes a dark from a bright color, but not the hue of a color. A specific color is detected by using all the three LEDs rapidly cycling. In this mode, the sensor can recognize these color values: black, blue, green, yellow, red, white, brown or none if the sensor does not recognize any specific color.

Gyro Sensor: it is a gyroscope that measures the degree of rotation of a robot with respect to its starting position and its rotational speed. The sensor returns positive or negative values if the robot is rotating respectively clockwise or counterclockwise. The angle of rotation is determined with an accuracy of +/- 3 degrees. The speed is measured as the degrees per second, with a sample rate of 1 kHz, allowing to detect a speed up to 440 degrees/second. This sensor is not a gyrocompass, i.e., it computes only the direction of the robot with respect to the direction it had when it started moving, but it does not compute the geographical direction of the robot.

Touch Sensor: this sensor consists of a button and detects if this button has been pressed or released.

(a) (b) (c)

Figure 5.3: LEGO Mindstorms EV3 Color Sensor (a), Gyro Sensor (b) and Touch Sensor (c) (Source: shop.lego.com).

5.2 Software

LEGO provides a free software for programming the robots. This software has an intuitive graphic interface, but it is not enough formal for our purpose, since it hides the real programming implementation, and besides, it requires Windows or Mac proprietary operative systems.
Fortunately, LEGO also releases the source code of the Linux distribution installed on the Brick, so that every one can modified it for his purpose. Two

open source communities developed these two interesting alternatives to the original operative system:

- *leJOS* is mainly based on the original Linux distribution of LEGO, modified in such a way to offer fully support for *Java* programming language.
- *ev3dev* is a more general purpose *Debian* Linux-based operative system, where all the EV3 sensors and motors are coded as kernel modules, hence they can be used simply by running a terminal command. This approach is flexible and many programming languages can be used on the ev3dev operative system. In particular ev3dev community offers full maintenance for *C++*, *Python* and *Nodes.js*, while the support for some other language, such as *Go*, depends on the contribute given by other users.

We used the ev3dev operative system [1] for its versatility and Python 3 as programming language, since it offers many advantages. First, Python is concise, with a simple syntax often very close to pseudo code. Then, Python is an interpreted language, thus it allows to test quickly a code typing it on the terminal. Finally, it supports the object-oriented programming, allowing to code with an high level of abstraction.

5.3 Implementation of the Theoretical Model

In this section we show how the theoretical model has been realized in practice, and how the real robots capabilities have been implemented exploiting the available motors and sensors.

5.3.1 Move on a Graph

A robot uses two wheels to move, each one connected to a Large Motor. The distance between these wheels is small so that the robot can perform small rotations. A third smaller wheel, not connected to any motor and with a little friction, is used to balance the robot. The robot dimensions are more or less $150 \times 165 \times 125$ mm.

A robot follows a link (line) implementing a *PID controller*, exploiting the Color Sensor and the motors of the wheels. In a machine a Proportional Integral Derivative (PID) controller continuously computes an error value, as

the difference between a target value and a measured value, and it corrects the action of the machine in order to reduce this error, according to the following equation [7]:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{5.1}$$

where $u(t)$ is the action of the machine at time t; $e(t)$ is the error computed at time t; K_p, K_i, and K_d are the proportional, integral and derivative terms multiplied respectively by the current error, the integral of the error over time and the derivative of the error.
Determining the best values for the K_p, K_i, and K_d coefficients, called *tuning*, is not trivial, however for our goal we obtained good results using just the proportional coefficient ($K_d = K_i = 0$), i.e., we implemented a Proportional controller. In our context, we drew black lines over a white background, so that the robot distinguish them, since black color has a very low reflection value, let us call it m_b, while white has an high reflection value, m_w. The robot is positioned in such a way that the Color Sensor is exactly over the border of a black line. The reflection value returned by the Color Sensor is based on the light measured in a neighbor circle of about 1 cm of diameter. If the robot moves following the line the center of the Color Sensor remains over the border of the line, thus the value that the sensor retrieves in this situation is the target measure, m, used by the proportional controller and it is equal to $m = (m_b + m_w)/2$. The error $e(t)$ is computed as $e(t) = m - m(t)$ where $m(t)$ is the measure obtained at time t. If the robot is following the internal border of a line (i.e., white is at the right and black is at the left) and it moves too to the right, the light it measures is more affected by the white color than by the black one, so $m(t) > m$ and $e(t) < 0$. Vice-versa if the robot moves too to the left $e(t) > 0$. The action of the controller is executed by the two Large Motors and consists of subtracting $e(t)$, multiplied by the opportune coefficient K_p, from the speed of the right wheel and summing up it to the speed of the left wheel, so that the robot continuously turns correcting its direction.
The proposed solution works regardless of the shape of the ring, provided that there are no sharp turns and the robots do not move too fast.

The robot has to detect when it reaches a node. A simple solution is to truncate a black line adding a colored marker that the robot identifies with the Color Sensor. In particular we marked only the special node $\otimes$ with a yellow strip, while the others have not been marked, i.e., when the line finishes the robot finds the white of the background. The robot knows it has reached a marker, and so a node, maintaining a list of the last colors sampled

by the Color Sensor and considering its median value. Both the yellow and the white colors have an high reflection value. Thus, when the line finishes and the Color Sensor is over the white or yellow color, the robot stops as soon as the median value of the colors sampled is over a certain threshold. When this happens, the robot exploits the Color Sensor in color mode, that allows to determine with more precision if the color is yellow or white.

When a robot reaches a node, it has to stop, go on or change its direction. We made many experiments before finding a design of a node allowing the robot to accomplish these tasks. The rotation is the most challenging operation, since a robot on a line reaches the other line only if it moves accurately. At the beginning we equipped a robot with the Gyro Sensor, that seems to be the natural choice to turn a robot. In theory a robot could know how much it has to rotate to reach another line, and the gyroscope should help it indicating when its degree of rotation is 180 degrees. In practice this solution does not work, because it assumes that the degree of rotation is 0 when the robot is moving perfectly straight on but determining when a robot is straight is not trivial. Due to the proportional controller a robot applies continuously small changes of direction, thus even if it is moving on a straight line we cannot know exactly when it is perfectly right. This consideration and the fact that, anyway, the gyroscope sensor does not have an high accuracy (+/- 3 degrees) make this sensor useless for our goal.
Fortunately, the Color Sensor can also be used to rotate over a node, provided that the node is designed properly (see Figure 5.4c). First the robot moves exploiting the proportional controller by considering that the black is to the left, i.e., it moves on the external border of a line (see Figure 5.4a). When the robot reaches a marker it is over a node. A robot crosses a marker applying a small movement until the Color Sensor, in reflection mode, measures a low value when it is again over a black line. To rotate over a node the robot uses again the proportional controller but reversing the positions of the black and of the white, i.e., it moves on the internal border of the black line (see Figure 5.4b).

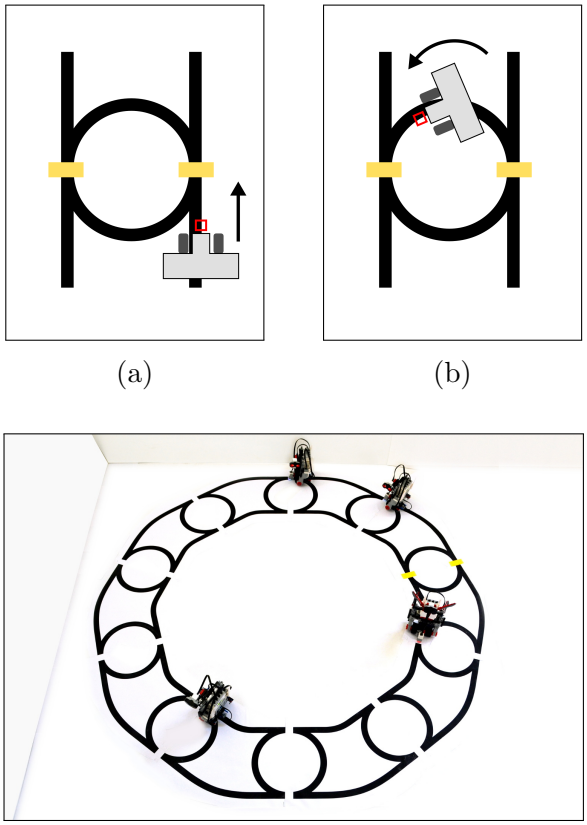


Figure 5.4: A robot (in gray) uses the Color Sensor (red square) to go straight on (a) or to rotate over a node (b). A ring of ten nodes used in our tests (c).

5.3.2 Robot Detection

To prevent a collision, an honest robot has to distinguish another honest one at the neighboring node or the malicious robot M, if it is located at the successive node. To detect a close robot at the same node, a robot can use the Infrared or the Ultrasonic Sensors. The Ultrasonic Sensor is accurate (+/- 1 cm) while the Infrared one is not precise. Moreover, our tests showed that the Infrared Sensor is much more subjected to interferences with respect to the Ultrasonic one. In practice, if two robots are at the two opposite markers of a node and they are looking with the Infrared Sensor the interference is so high that the measures could be inconsistent and one robot could not notice the presence of the other. Thus, we used the Ultrasonic Sensor to detect a close robot. We equipped each robot with one Ultrasonic Sensor, connected

to a Medium Motor, so that the robot rotates this sensor to obtain a view of more or less 180 degrees. The robot that is moving on a line looks in front of it to detect a possibly stopped robot at its destination node. Once reached a node marker, a robot looks sideways too see if another robot is located at this node.

Recognizing the malicious robot M is more challenging. The Ultrasonic Sensor cannot accomplish this task, in fact if a robot uses it to look at the neighboring node, it detects the presence of a robot, but it cannot distinguish an honest robot from the malicious one, even if M is connoted with a particular shape. We have also to consider that the agent M is too distant from an honest one to be detected with the Color Sensor.
Conversely, the Infrared Sensor is suitable for this task having the possibility of measure the intensity of a light generated by a beacon. In detail, M is equipped with the Infrared Beacon and emits an infrared light in one specific frequency, while the honest robots use the Infrared Sensor in listener mode. Since the honest robots only measure and do not emit a light the Infrared Sensors do not interfere. When M is at a node it must be detected by the robots moving in two opposite directions. This consideration is not trivial since the proposed solution works only if the beacon has a particular orientation with respect to the Infrared Sensor, but the beacon cannot be simultaneously well oriented to the Infrared Sensors of two robots moving in opposite directions. To overcome this problem first we equipped the robot M with two beacons oriented in opposite directions, but this solution does not work because the two beacons interfere, even if they emit a light on two distinct channels. Hence, we devised another solution that works properly: M has one beacon connected to a Medium Motor and it continues to rotate the beacon, so that the agents detect it in whichever direction.

5.3.3 Robot Communication

A robot has to apply a face-to-face communication, i.e., it communicates only with the other robots at the same node. Although establishing a communication between two or more robots can be achieved easily with a Bluetooth connection, communicating with only the closest robots is problematic and we have not found a solution for this task. As a first attempt we tried to exploit the Received Signal Strength Indicator (RSSI) of the Bluetooth device. The idea is that a robot detects a strong Bluetooth signal when another robot is at the same node and it communicates with the robot found. To obtain the RSSI of a Bluetooth device we found only a solution related to

the *hcitool* command available on Linux. Unfortunately, the values obtained in this way are useless: the RSSI detected for a robot at a same node can be equal to the one for a very distant robot. The poor results obtained and the lack of documentation in this topic led us to give up this solution.
Neither the infrared light can be used to establish a communication between robots, as we explain in the following. Suppose each robot is equipped with an Infrared Sensor and an Infrared Beacon. When they are at a node, they could communicate emitting an infrared signal on the same channel with the Infrared Beacon and measuring the other signals with the Infrared Sensor. Given that the robots are synchronous, they could distinguish the states of the others, because a robot in a particular state emits a signal in a specific time interval. In a sense, this solution could be an implementation of [23], where the state of a robot is expressed with different colored lights, in our case with infrared light, but in this context it does work. In fact, the beacon can be turned on or off only manually, therefore a robot cannot emit an infrared signal in a specific time interval during the protocol execution.

From the previous considerations we claim that our robots cannot apply a face-to-face communication. We used alternative kinds of communication explained in the next section.

5.4 Tests and Technical Limitations

We had available four EV3 robots, thus we used one robot as the malicious one, and the others as the honest ones. We tested the unorieted protocol (Algorithm 3), that also solves the oriented case in which all the agents move in the same direction.
When it was necessary, we slightly changed the theoretical protocol for practical reasons, without compromising its correctness. The major modification concerns the change of direction of the robots. In the theoretical protocol a robot can meet a stopped robot at a node and later it can reverse its direction, whereas in the practical protocol the stopped robot moves in place of the other one, to prevent the two robots overtake over a node that could result in a collision.

We implemented two different solutions that differ for the communication used.
The first proposed solution requires a wireless network in which the robots use their Bluetooth adapters to connect to a central server, that simplifies the communication between the robots and the detection of M. Every time

an honest robot reaches a node it notifies the server. The server returns to an honest robot the states of the other robots at the same node, and it communicates to the robot if it is blocked by M or not. Conversely, the robot M can query the server at any time, obtaining all the information related to the honest robots. The drawback of this approach is that the robots are no more autonomous.

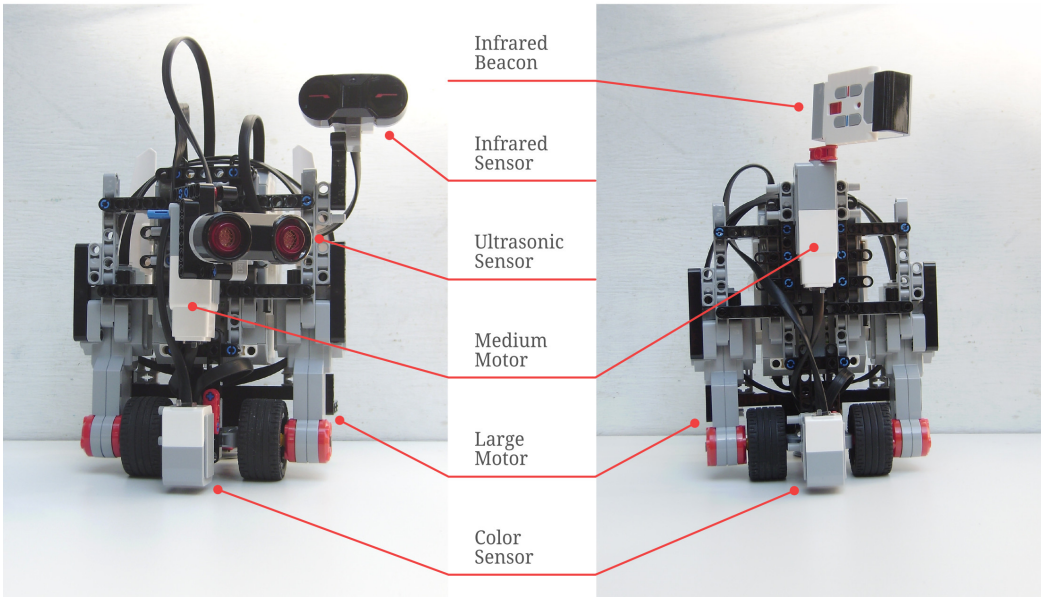


Figure 5.5: An honest robot (to the left) and the malicious one (to the right) and the devices used to implement their capabilities.

The second solution that we propose is completely distributed: a robot determines the state of another one according to its position or its movement, similarly to [16]. In this solution the sensors play a more important role since the Ultrasonic Sensor is fundamental to deduce the state of a robot, and the Infrared Sensor is used to detect M (Figure 5.5 shows the robots used in the distributed solution). Unlike the solution based on the server, that can be extended for more than three robots, in the distributed solution we assume that the robots are exactly three, since some situations, such as the communication of the *moved* and of the *parity* variables (not necessary with only three robots) cannot be implemented.
In the following, we list the major conventions used in the distributed solution to realize an undirected communication between the robots. First, a robot in state *STOPPED* stops at a marker, so that when a robot in state *INITIAL* or *MSG* detects the presence of a robot before meeting a marker

it knows this is a *STOPPED* robot. Moreover, a robot in state *STOPPED* distinguishes a *MSG* robot from an *INITIAL* one because the *MSG* robot waits a clock stopped at a node (this fact does not compromise the correctness of the algorithm since no other robot after the *MSG* one reaches the robot in state *STOPPED*). Finally, a robot in state *STAR* is distinguishable from a robot in state *HEAD* because they occupy two different positions over a node.

We performed many tests obtaining that in the majority of the cases the robots achieve the rendezvous as expected. When this does not happen, the failure is caused by one or more of the following motivations.
The Color Sensor is extremely important for the success of the protocol but it is strongly affected by the environment enlightenment. The graph surface has to be evenly illuminated to ensure that a same color remains equal regardless of the point where it is sampled, in particular the white one that reflecting more light is very susceptible to light variations. Furthermore, even if the robots are equipped with the same type of sensors, their functioning is not always equal. Particularly, different Color Sensors could measure slightly different values. To prevent these issues, before running the protocol, every robot has to calibrate the black, white and yellow colors values, that change according to the robot and the external enlightenment.
If the graph surface is not perfectly flat, this affects the light reflected and it can also cause that a robot moves improperly, in the worst case exiting from a line.
Both the Ultrasonic and the Infrared Sensors are subjected to interferences, that are higher when two sensors are close and located one in front of the other, so we avoided as much as possible to create this situation. Also the presence of external objects, such as walls, affects the sensors values, in particular the infrared one. For this reason the experiments must be executed in an environment without obstacles.

Chapter 6

Conclusions

In this thesis, we have surveyed the most recent works related to the rendezvous problem of autonomous mobile robots in the distribute algorithms area, and we have pointed out that the obtained theoretical results have scarcely been tested in a real scenario. Unlike these works, we have proposed novel solutions for the rendezvous problem that have been applied to real robots.
For our purpose we have considered the model of [24] with k honest robots and one malicious robot, and we have extended it to the synchronous case. We have proposed solutions to the rendezvous problem for $k \geq 2$ robots in the oriented and the unoriented ring. These solutions are efficient in terms of memory, execution time and number of moves required.
We have illustrated how our theoretical protocols can be applied to real general-purpose LEGO Mindstorms EV3 robots. We have focused on the methods used to implement the robot capabilities, on the difference between the real and the theoretical protocols and on the technical limitations we faced.

As a future work, it would be interesting to test with real robots also asynchronous protocols, that are more complicated to simulate than the synchronous ones.
Additionally, we are also interested in testing our protocols with more robots and we are studying how to extend the proposed solutions to other topologies.
Another step could be trying to implement with real robots the theoretical protocols for the rendezvous problem in the plane.
Finally, the main issue we faced in the practical implementation is the face-to-face communication between robots, that apparently is not realizable with our robots. It would interesting to check if there exists some hardware that

allows to apply this kind of communication, such as a programmable infrared beacon that can be turned on only when it is useful for communicating, and turned off otherwise, in order to avoid interferences. Alternatively, we could try to modify the LEGO Infrared Beacon to meet these requirements.

Bibliography

[1] ev3dev operative system, 2016. http://www.ev3dev.org/.

[2] LEGO Mindstorms EV3, 2016. http://www.lego.com/en-us/mindstorms/products/31313-mindstorms-ev3.

[3] Noa Agmon and David Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM Journal on Computing*, 36(1):56–82, 2006.

[4] Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*, volume 55. Springer Science & Business Media, 2006.

[5] Hideki Ando, Izumi Suzuki, and Masaru Yamashita. Formation and agreement problems for synchronous mobile robots with limited visibility. In *Intelligent Control, 1995., Proceedings of the 1995 IEEE International Symposium on*, pages 453–460. IEEE, 1995.

[6] Hideki Ando, Yoshinobu Oasa, Ichiro Suzuki, and Masafumi Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *Robotics and Automation, IEEE Transactions on*, 15(5):818–828, 1999.

[7] Karl J Astrom. Pid controllers: theory, design and tuning. *Instrument society of America*, 1995.

[8] Chanderjit Bajaj. The algebraic degree of geometric optimization problems. *Discrete & Computational Geometry*, 3(2):177–191, 1988.

[9] Lali Barrière, Paola Flocchini, Pierre Fraigniau, and Nicola Santoro. Can we elect if we cannot compare? In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 324–332. ACM, 2003.

[10] Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Election and rendezvous in fully anonymous systems with sense of direction. In *SIROCCO*, pages 17–32, 2003.

[11] George A Bekey. *Autonomous robots: from biological inspiration to implementation and control*. MIT press, 2005.

[12] Kálmán Bolla, Tamás Kovacs, and Gábor Fazekas. Gathering of fat robots with limited visibility and without global navigation. In *Swarm and Evolutionary Computation*, pages 30–38. Springer, 2012.

[13] Jérémie Chalopin, Emmanuel Godard, Yves Métivier, and Rodrigue Ossamy. Mobile agent algorithms versus message passing algorithms. In *Principles of Distributed Systems*, pages 187–201. Springer, 2006.

[14] Jérémie Chalopin, Yoann Dieudonné, Arnaud Labourel, and Andrzej Pelc. Rendezvous in networks in spite of delay faults. *Distributed Computing*, pages 1–19, 2015.

[15] Sruti Gan Chaudhuri and Krishnendu Mukhopadhyaya. Leader election and gathering for asynchronous transparent fat robots without chirality. *arXiv preprint arXiv:1208.4484*, 2012.

[16] Mark Cieliebak. Gathering non-oblivious mobile robots. In *LATIN 2004: Theoretical Informatics*, pages 577–588. Springer, 2004.

[17] Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Solving the robots gathering problem. In *Automata, Languages and Programming*, pages 1181–1196. Springer, 2003.

[18] Jurek Czyzowicz, Leszek Gasieniec, and Andrzej Pelc. Gathering few fat mobile robots in the plane. *Theoretical Computer Science*, 410(6): 481–499, 2009.

[19] Jurek Czyzowicz, Andrzej Pelc, and Arnaud Labourel. How to meet asynchronously (almost) everywhere. *ACM Transactions on Algorithms (TALG)*, 8(4):37, 2012.

[20] Shantanu Das. Mobile agent rendezvous in a ring using faulty tokens. In *Distributed Computing and Networking*, pages 292–297. Springer, 2008.

[21] Shantanu Das. Mobile agents in distributed computing: Network exploration. *Bulletin of EATCS*, 1(109), 2013.

[22] Shantanu Das, Matúš Mihalák, Rastislav Šrámek, Elias Vicari, and Peter Widmayer. Rendezvous of mobile agents when tokens fail anytime. In *Principles of Distributed Systems*, pages 463–480. Springer, 2008.

[23] Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masaru Yamashita. The power of lights: Synchronizing asynchronous robots using visible bits. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 506–515. IEEE, 2012.

[24] Shantanu Das, Flaminia L Luccio, and Euripides Markou. Mobile agents rendezvous in spite of a malicious agent. In *Algorithms for Sensor Systems*, pages 211–224. Springer, 2015.

[25] Gianluca De Marco, Luisa Gargano, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Ugo Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theoretical Computer Science*, 355(3):315–326, 2006.

[26] Anders Dessmark, Pierre Fraigniaud, Dariusz R Kowalski, and Andrzej Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, 2006.

[27] Yoann Dieudonné and Andrzej Pelc. Deterministic network exploration by a single agent with byzantine tokens. *Information Processing Letters*, 112(12):467–470, 2012.

[28] Yoann Dieudonné, Andrzej Pelc, and David Peleg. Gathering despite mischief. *ACM Transactions on Algorithms (TALG)*, 11(1):1, 2014.

[29] Yoann Dieudonné, Andrzej Pelc, and Vincent Villain. How to meet asynchronously at polynomial cost. *SIAM Journal on Computing*, 44 (3):844–867, 2015.

[30] Krzysztof Diks, Pierre Fraigniaud, Evangelos Kranakis, and Andrzej Pelc. Tree exploration with little memory. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 588–597. Society for Industrial and Applied Mathematics, 2002.

[31] Paola Flocchini and Nicola Santoro. Distributed security algorithms for mobile agents. *Mobile Agents in Networking and Distributed Computing*, pages 41–70, 2012.

[32] Paola Flocchini, Evangelos Kranakis, Danny Krizanc, Flaminia L Luccio, Nicola Santoro, and Cindy Sawchuk. Mobile agents rendezvous when tokens fail. In *Structural Information and Communication Complexity*, pages 161–172. Springer, 2004.

[33] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, 337(1):147–168, 2005.

[34] Pierre Fraigniaud and Andrzej Pelc. Decidability classes for mobile agents computing. In *LATIN 2012: Theoretical Informatics*, pages 362–374. Springer, 2012.

[35] Tamer Inanc and Huan Dinh. A low-cost autonomous mobile robotics experiment: Control, vision, sonar, and handy board. *Computer Applications in Engineering Education*, 20(2):203–213, 2012.

[36] Ahmed K Noor. Moving on their own: technology is advancing toward a future supported by a workforce of autonomous, mobile robots. *Mechanical Engineering-CIME*, 130(11):26–32, 2008.

[37] Edwin Olson. A primer on odometry and motor control, 2004.

[38] Attila Pásztor. Gathering simulation of real robot swarm. *TEHNICKI VJESNIK-TECHNICAL GAZETTE*, 21(5):1073–1080, 2014.

[39] Giuseppe Prencipe. Impossibility of gathering by a set of autonomous mobile robots. *Theoretical Computer Science*, 384(2):222–231, 2007.

[40] Giuseppe Prencipe. Autonomous mobile robots: A distributed computing perspective. In *Algorithms for Sensor Systems*, pages 6–21. Springer, 2013.

[41] Nicola Santoro. *Design and analysis of distributed algorithms*, volume 56. John Wiley & Sons, 2006.

[42] Ichiro Satoh. Mobile agents for ambient intelligence. In *Massively Multi-Agent Systems I*, pages 187–201. Springer, 2004.

[43] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[44] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

[45] Ying Tan and Zhong-yang Zheng. Research advance in swarm robotics. *Defence Technology*, 9(1):18–39, 2013.