ULB UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences
Département d'informatique

IRIDIA
Institut de Recherches Interdisciplinaires
et de Développements
en Intelligence Artificielle

Gesture Control for Swarms of Robots

Gaëtan Podevijn

Mémoire présenté sous la direction du **Prof. Marco Dorigo**
sous la co-direction de **Dr. Rehan O'Grady**
en vue de l'obtention du grade de Master en Sciences Informatiques
Année Académique 2011-2012

Abstract

Interacting with a swarm of robots is an essential need for a human who wants to benefit from swarm robotics. This interaction is more complex than guiding a simple agent because in a swarm, robots all have a different frame of reference. To date, there are only a few studies that focused on human-swarm interaction. In this work, a complete framework for interacting with several swarms is developed. It offers the essential robot behaviour needed for the user to control the swarms. This control must be precise in order to guide the groups accurately in the environment. The robots also have to work collectively so that the user can focus on groups instead of on individuals. The interaction system used in this thesis is based on gesture recognition but can be easily adapted to any other kind of control devices. We proved the feasibility of the frramework by both simulation and real world experiments. We also developed a set of statistical tools to measure the usability of human-swarm interaction systems.

Résumé

L'interaction avec un essaim de robots est une nécessité pour qu'un humain puisse bénéficier des avantages de la robotique collective. Diriger un tel nombre d'individus est une tâche plus compliquée que celle consistant à guider un seul agent. Ceci est principalement dû au fait que les robots ont chacun un référentiel différent et que l'utilisateur ne peut pas s'adapter à chacun d'eux. A notre connaissance, très peu d'études ont été effectuées sur l'interaction entre un homme et un groupe de robots. Dans ce travail, un système d'interaction permettant de diriger plusieurs essaims a été développé. Il fournit un ensemble d'actions que les robots doivent être en mesure d'exécuter afin de donner à l'utilisateur la précision nécessaire ainsi que les avantages de la robotique collective. Des résultats positifs ont été obtenus en effectuant des expériences à l'aide d'un simulateur, mais aussi en conditions réelles. Des outils statistiques adaptés à l'étude de l'utilisabilité de systèmes d'interactions ont également été développés, dans le but de fournir aux chercheurs la possibilité de mesurer et d'analyser l'utilisabilité d'un tel système.

Je dédie ce travail à la mémoire
de mon grand-père, François Podevijn.

Remerciements

Je remercie le Prof. M. Dorigo d'avoir accepté d'être le directeur de ce mémoire et d'avoir mis à ma disposition tout le matériel nécessaire afin de travailler dans les meilleures conditions.

Je voudrais remercier tout particulièrement le Dr. Rehan O'Grady de m'avoir supervisé durant toute cette année. Sa motivation communicative et ses conseils avisés m'ont permis d'avancer durant les périodes les plus creuses.

J'exprime toute ma gratitude aux autres membres de mon Jury, le Prof. M. Birattari et le Prof. H. Bersini pour le temps qu'ils passeront à lire ce document.

Je n'oublierai pas Youssef S. G. Nashed, doctorant à l'Université de Parme, avec qui j'ai eu l'opportunité de collaborer durant trois mois.

Que soient remerciés tous les membres d'IRIDIA que j'ai eu l'occasion de rencontrer durant cette année. Leur aide et leur bonne humeur ont joué un grand rôle dans la réalisation de ce travail.

Un très grand merci à mes parents et ma soeur pour leur soutient, durant cette année bien sûr, mais également durant toute la durée de mes études. Je tiens aussi à remercier mon oncle pour ses relectures et l'aide qu'il m'a apporté concernant la rédaction de ce document.

Enfin, le plus grand des mercis à Caroline pour la patience dont elle fait preuve, et pour le bonheur qu'elle me procure quotidiennement.

Contents

Chapter 1

Introduction

In our daily life, we regularly face problems we cannot solve by ourselves. We ask for help and work *together*, think *together* and hopefully, solve problems *together*. This group work can also be observed in nature. Bees, ants and termites are examples of insects that work collectively. Compared to the magnitude of their work, these agents are physically limited. For example, termitaria may reach up to 9 meters high and up to 30 meters in diameter, while a single termite's size is from 2 mm to 8 mm. Moreover, most of the termites, especially workers termites, are blind. Their collective behaviour allows them to achieve very complex results that transcend their individual limitations.

Based on these observations, scientists, and more specifically roboticists, were inspired by the collective behaviour of physically simple agents. They tried to adapt it to artificial individuals. They imagined that by following some simple rules, a group of physically limited robots could resolve some relatively complicated tasks, that is, tasks that each individual could not resolve by itself. This idea is in stark contrast with more classical robotics approaches, where each agent is a very complex unit.

Since the early 2000s, two successful European projects have been conducted on swarm robotics. The Swarm-Bots project [1] and the Swarmanoid project [2] were funded to the tune of multi-millions of euros by the Future and Emerging Technologies programme (FET).

These projects successfully demonstrated the ability for physically simple artificial agents to work collectively and reach complex results in a human environment. However, even after a decade of research, few studies have focused on the interaction between a human and the swarms. So far, swarms are able to work autonomously

1. http://www.swarm-bots.org
2. http://www.swarmanoid.org

on a specific task in a specific environment. The autonomy is limited to the task the robots are executing, they do not take higher level decisions. If we want swarm robotics to be useful for humans one day, they must be able to give high-level orders to the robots. We can make an analogy with a sergeant who gives orders to his soldiers. Once the soldiers receive the order, they execute it without requiring anything else from their sergeant. In a swarm robotics system, a human, like the sergeant, must be able to send its requests to the robots. Once the robots received an order, they can start working autonomously on the given task.

Controlling a single agent with powerful perception and motion sensors is relatively simple. It can be done by adapting the human perception with respect to the robot perception. For swarms however, this method is not feasible because all the robots belonging to a swarm have different perceptions of the environment and different frames of reference. Therefore, the human cannot adapt his own perception to the ones of the thousand of robots. So, how may a human control all of these robots in a complex environment? Clearly, controlling each single robot is inefficient because of the large number of them. Choosing a leader is not scalable because if this leader encounters some issues (low-level battery, physical crash, . . .) the rest of the group would cease to function. Consequently, we need to provide the human operator with a way to send its instructions to a very large set of robots in such a way that every robot knows what to do, without the need for the operator to interact with each single agent.

In this thesis, we have focused on the entry level task of moving swarms of robots to task locations. Robotic movement is important because before being able to carry out a task, the robots must be able to get to the site of the task. The movement of robotic swarms gives us a self-contained activity that we can use to take some important first steps into the field of human swarm interaction. Over the course of this thesis we developed a framework that allowed a human operator to divide a robotic swarm into sub-swarms and send these sub-swarms to various task location sites.

1.1 Goal of This Work

The goal of this work was to address the scientific and technical challenges associated to the development of a tool that allows a human to interact with swarms formed by a large number of autonomous robots.

The main contribution of this work is the conceptualisation, design and development of several commands that are necessary for the operator to control swarms in a complex environment. We conceived these commands in such a way that the

user can manipulate the swarm as a single entity. This allows the user to have an abstraction of all the robots in the swarm. Consequently, the operator does not have to consider each single individual. To enable these commands, we also needed to create a low-level architecture that allows a human to interact with many robots at a time. Finally, we had to develop the interface between the operator and the robots, that is, the means by which the human actually send its queries. In this work, we chose gestures as an intuitive means for controlling the robots. We confirmed the feasibility of our system by conducting experiments in which we asked volunteers to control swarms of simulated robots.

A second contribution of this work is the development of a framework used to conduct usability measures for comparing different interaction modalities. We defined a detailed and rigorous procedure to study the efficiency of these systems. Based on the results of our experiments, we proposed statistical tests that can be used for comparing and studying human swarm interaction techniques.

1.2 Outline of This Thesis

In Chapter 2 we present the background necessary for this work. Then, in Chapter 3 we develop the swarm robotics part of the system. In the following Chapter 4 we present the tools created to achieve the gesture recognition part. Chapter 5 is devoted to the usability experiments in which we propose a complete methodology to study the ease of manipulation of the system. Next Chapter 6 presents statistical tools that can be useful to analyse the usability difference between several interaction devices. Finally, we summarize in Chapter 7 the work achieve so far and discuss future potential studies.

Chapter 2

Background

In this chapter, we present the underlying fundamentals of this work. First, we present Swarm Intelligence and Swarm Robotics. Afterwards, we review the notion of human computer interaction and we see the novel approaches of this field.

2.1 Swarm Intelligence

Swarm Intelligence, commonly abbreviated as SI, can be loosely defined as being a sub-field of artificial intelligence. A swarm intelligence system is made up of relatively simple individuals interacting locally with each other and following simple behavioural rules. From these interactions, a global behaviour can emerge. In a swarm intelligent system, these individuals, also called agents, have a restricted capacity of reasoning and representation. Their capacity of resolving a complex task lies in the collective interaction of the individuals. Each agent takes its own decisions based on the local information.

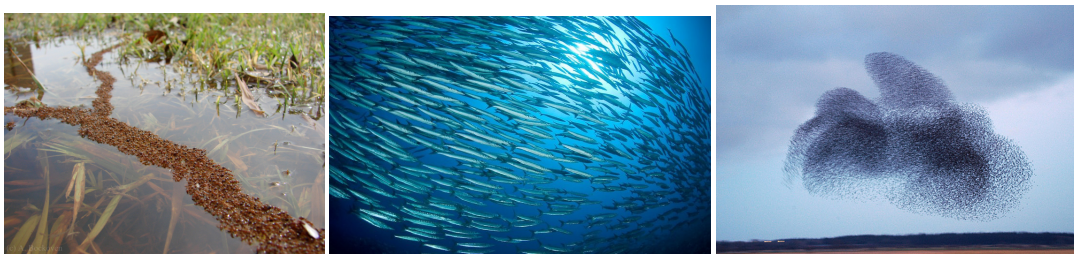
The term swarm intelligence has been proposed by Beni and Wang in 1989 [4]. SI systems are inspired by the collective behaviour of groups of animals or of social insects. Ants colonies [17], fish schooling [25] bird flocking [54] are all examples of natural collaborative systems (see figure 2.1). The four main characteristics of a social insect colony are:

Flexible: their work is not perturbed by internal or external changes.

Robust: even if some agents fail during a task, the others can still reach their goal.

Decentralized: there is no leader agent in the colony.

Self-organized: the resulting solution emerges from local interactions and simple behaviour, instead of being predefined.



(a) (b) (c)

Figure 2.1 : (a) a path on water formed by fire ants (source: `http://6legs2many.wordpress.com`). (b) school of fish (source: `http://scottpenny.smugmug.com`). (c) birds flock (source: `http://armedwithvisions.com`).

So far, we defined swarm intelligence quite broadly. A more commonly and rigorous definition is given by Bonabeau *et al.* [6]: *"Any attempt to design algorithms distributed problem-solving devices inspired by the collective behaviour of social insect colonies and other animal societies"*.

Applications of swarm intelligence are vast: optimization algorithms [19], data mining [24], network routing [11] and, the one that interests us the most, swarm robotics [18].

2.1.1 Swarm Robotics

The term *swarm robotics* emerged from the use of robots to study swarm intelligent systems [14]. This sub-field of swarm intelligence focuses on the study of the techniques used to emerge artificially a global and collective behaviour from a large number of physically and simple embodied agents. These agents interact with each other and their environment [31]. The following five criteria have been proposed as a working definition of a swarm robotics system [31]:

Autonomy: it is required that the robots are autonomous and do not depend on any other kind of agent. They must be able to interact with their environment.

Large number: even if we do not explicitly define the minimum number of robots belonging to a swarm, a large number of units is required in order for cooperative behaviour to occur.

Limited capabilities: the robots of the swarm should be relatively inefficient on their own. The tasks that swarm robotics systems intend to solve are too difficult for a unique entity.

Scalability and robustness: the deployment of new agents in the system should improve the efficiency of the handling of the task. On the other hand, removing

some entities should not cause dramatic failure.

Distributed coordination: the robots should only communicate with local interaction and should sense a limited part of the environment.

Thanks to the relatively low cost of robots, swarms of robots will be able to be used in industry. Şahin identifies in [14] some applications. The distributed characteristic of such systems makes their use well-suited for tasks that handle localization in large spaces, like finding the place of a specific problem (see figure 2.2(a)). Other good opportunities to use the collaborative behaviour of swarm are tasks that are too dangerous for human beings. An example of such a task is the cleaning of a mine field (see figure 2.2(b)). Eventually, we saw that robustness is required in a swarm of physical agents. If robots are communicating with each other as in a very large network, a node loss in the network should not prevent the system from continuing to operate (see figure 2.2(c)).

(a) (b) (c)

Figure 2.2 : (a) Robots could explore unknown areas (source: `http://www.fastcompany.com`). (b) Instead of human beings, one could imagine a swam of robots exploring a mine field (source: `http://www.cyberpresse.ca`). (c) A network of robots.

2.2 Robot Hardware

Robots used in this work were built during the Swarmanoid project [44]. The Swarmanoid project had as objectives to propose a new distributed robotic system compound of heterogeneous robots. The robots used are called *foot-bots* (see figure 2.3). They can move on the ground and can self-assemble in order to form larger and powerful entities.

They are built on a multi-processor architecture running under a Linux operating-system. A main processor manages the heavy computation, such as vision processing and other higher-level control while other micro-controllers focus on the sensors embedded on the robots. The main processor is a 533 MHz i.MX31 ARM 11, with

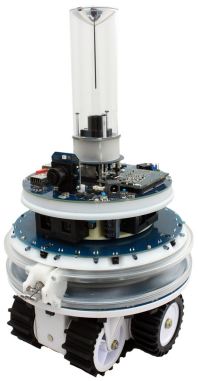


Figure 2.3 : A foot-bot.

128MB of main memory and 64MB of FLASH to store data. There is a USB 2.0 controller to access the system and they fit a 802.11g wireless network controller to either access the system or to let the robots communicate with each other.

A second mean of communicating for the robots is the range and bearing sensor which allows them to "feel" other robots in a 3D environment. This sensor has a precision of 10 cm up to 5 m and uses IR rays and radio to communicate.

The size of the foot-bots is 28 cm high and 13 cm of diameter. They have a differential drive system which allows them to go up to 30 cm/s. To allow self-assembling, they use a gripper located above the base module. There is a 12 LEDs ring and a light beacon which allows foot-bots to communicate using colour detection. There are two distance sensors for short and long ranges. Two cameras are used for top detection and an omnidirectional one for environment detection. A torque sensor allows the robots to measure the force applied on themselves. Finally, an RFID reader and writer is located on the bottom of the foot-bots.

2.3 Simulator

In almost all scientific fields, simulations are a crucial step during the elaboration of new concepts. Simulations are intended to mimic real systems and are used to retrieve useful data, as if we would have access to the real environment. The same simulations can be performed multiple times having either tuned some parameters or having changed all the implementation of the solution in itself, allowing us to compare the results between the different solutions.

Robotic and swarm robotics also extensively use simulations. There is a number of advantages of using simulations to create innovative robotic algorithms. First,

it allows the developers to rapidly test novel concepts without the need to use real robots. Simulators are also used to speed up the processing time, gaining a large amount of time. Moreover, building real robots is still expensive, especially if a lot of robots are required for the experiment. Using a simulator allows the use of as many robots as needed, from ten to thousands of them. In addition, unexpected situations such as robots crashing have no cost effect during simulations.

A perfect simulator would allow us to assure that if the simulations are correctly working, then real world executions of the algorithms would give exactly the same results. Unfortunately, this is not true. Indeed, what gives their advantages, such as the elimination of a few real parameters that are difficult to model also gives them their disadvantages. The same parameters that were removed, or even parameters that the designers did not think of make the real executions different from what was expected.

For this work, ARGoS (for Autonomous Robots Go Swarming) [51], a robotic simulator has been extensively used during all parts of the project. ARGoS has been developed during the Swarmanoid project to tackle the disadvantages of the other existing simulators [44]. The previous simulators focused either on scalability, that is, supporting a large amount of robots, or on flexibility, that is, supporting a high number of different robots. The result of Swarmanoid's simulator is a highly flexible and scalable multi-robot simulator.

Figure 2.4 depicts the architecture developed for ARGoS. The core idea of the architecture is the use of *modules* that are loaded at run-time and that are organized around the simulated space. This space contains the information about the entities that are present in the space such as their position and their orientation. The *controllers* interact with the simulated space through *sensors* and *actuators*.

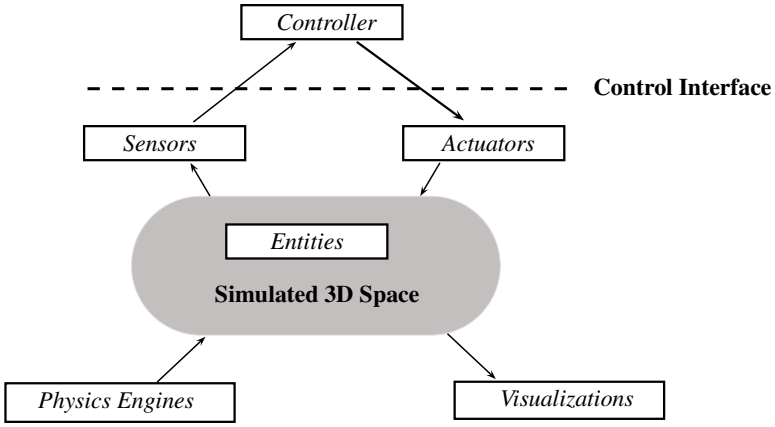


Figure 2.4 : ARGoS architecture. Source: [51]

The controller is the module that allows the user to create the robot's behaviour by

accessing both the sensors modules and the actuators through the control interface. This interface is the same on real robots, which allows the user to write the code only once. The two last modules are used to communicate with the simulated space: the sensors receive data about the environment, such as range and bearing data. The actuators allow the robot to interact with the space, for example, writing data in the range and bearing's payload.

The physics engines are responsible for updating the state of the entities present in the space (such as robots, walls and mobile or non-mobile objects). Multiple physics engines can be used in the same simulation, which is one of the most powerful features of ARGoS. The visualization modules render the simulated space. In ARGoS, three visualization modules exist. The first is an OpenGL-based graphical interface. The second uses POV-Ray to render a higher quality of what is happening in the environment. Finally, a text-based feedback can be used for post-simulation statistics.

All these modules can be created, modified and configured through an XML configuration file. This eliminates the necessity to compile the controller after each modification.

2.4 Human Computer Interaction

One of the many challenges in computer systems is the mean of interaction between them and the users in such a way that they can use the system with a minimum of instructions. This section retraces the history of the development of user interfaces and introduces a new paradigm, namely the Natural User Interface (NUI).

2.4.1 History

During the 1950s, computers were not able to perform multiple tasks at the same time and resources were very expensives. There was no "real" interaction as we know today, that is, the user and the system could not interact in real-time. Users had to deal with Batch interfaces (see figure 2.5(a)) which are non-interactive: users gave the input of the task which had to be done and the batch system outputted the results after several hours or even after days.

The 1960s introduced the Command Line Interface (CLI). People interact with a computer system through a keyboard to send their commands. CLI was a big step in human computer interaction because coupled with the improvements of computing power, users could interact with their system in near real-time. The disadvantage

of this system is that the syntax of the command is very strict. Users must enter the syntactically correct commands and memorise them. It takes a large amount of time to master the system and to become familiar with it.

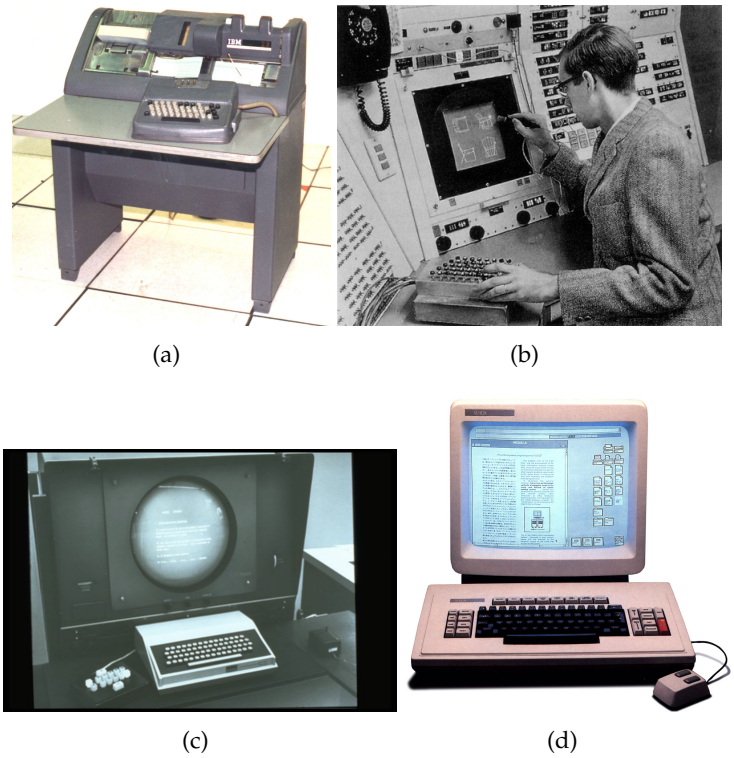
During the same decade, two big inventions were created. The first was produced by Ivan E. Sutherland during his PhD thesis. It is the pioneering system of the user interaction systems: Sketchpad Sutherland [55] (see figure 2.5(b)). This system used a light pen which allowed the user to draw on the screen. It was possible to select objects and to move them on the screen area. Ivan Sutherland received the Turing award in 1988 for his contribution, which is considered as being the first graphical user interface [10]. The second, more known and still used today is the mouse. It was developed at the Standford Research Laboratory by Doug Engelbart as part of his NLS project (oN-line System, see figure 2.5(c)) which was the first to use hyper-text links and a mouse as a pointing device [20].

The first machine that looks like everything we know today was the Xerox Star system (see figure 2.5(d)) developed by Xerox Corporation in 1981. It was the first personal computer using graphical user interface that we are now used to seeing [32].

2.4.2 Towards a New Paradigm

Graphical User Interface follows a paradigm called WIMP for Window Icon Menu Pointing device [59]. It has been developed in order to give the user a representation model of the system. With GUI, the users manipulate windows and buttons while icons are a good way to give a meaning for a specific command and menus order the available actions. A. Van Dam argues in [59] that the WIMP paradigm is becoming more and more inappropriate. The first reason is that the more complex an application becomes, the harder the learning phase to use the application becomes. The second issue raised by the author is that users spend too much time using the interface instead of the application. They have to navigate a lot in the interface before doing something useful.

Ron George and Joshua Blake go deeper in the analysis of WIMP. According to them, WIMP requires a too high cognitive load while interacting with the GUI. This can prevent some people with limited cognitive faculties from using these systems [22]. To address the previous issues, George and Blake came up with a new paradigm, namely OCGM. Instead of manipulating windows, buttons and menus, in the OCGM model, the users manipulate **O**bjects, **C**ontainers (sets of objects), **G**estures and **M**anipulations. The two lasts differ in the sense that with a manipulation, the object reacts immediately (for example: moving an object in the space) while with a gesture, the system waits until the end of the gesture to achieve a particular command. According to the authors, this new model allows the user to



(a) (b)

(c) (d)

Figure 2.5 : (a) is the IBM 26 working as a batch system (source: `http://www.tietokonemuseo.saunalahti.fi` on `webarchive.org`) (b) is the Sketchpad Sutherland in action (source: `http://www.mprove.de`). (c) is the oN-Line System created by Engelbart (source: `http://sloan.stanford.edu`). (d) shows a Xerox Star person computer. (source: `http://www.digibarn.com`)

interact with a system thanks to his cognitive skills developed during his early age. This tends to make the system more intuitive and natural to use.

This new paradigm allowed the emergency of Natural User Interfaces (NUI) which are based on natural human communication behaviours. With NUI, users can interact with the representation of an object in the system like they would interact with the same object in the real world. For several years now, we have seen several kinds of NUI systems emerge from video games companies. In 2006, Nintendo proposed a new video game console, the Wii, using a remote controller (Wiimote) that uses an accelerometer to detect positions in a 3D space. In 2010, Microsoft launched the Kinect device which allows a user to play video games using its body as a controller.

In the context of robot interaction, NUI clearly has advantages over more classical interaction devices. Classical devices require the user to learn their functioning. It can break down and the user needs to carry around it. On the other hand, if the

robots are able to understand the meaning of the usual human communication (i.e. language or gestures), then the user could interact with the robots like he would interact with other humans, in a natural way.

Chapter 3

Human-Swarm Interaction

3.1 Motivation

Swarm robotics, as we defined in section 2.1 focuses on the development of distributed algorithms for autonomous robots. These agents are capable of achieving specific tasks in a known or unknown context, communicating with each other and with their environment. A global behaviour emerges from a large number of simple entities.

These entities however, are not fully autonomous. For a swarm to be useful to a human, the human has to be able to tell the swarm what task to execute and where to execute it. Other situations in which controlling the robots is useful are when the interactions between the robots results in unpredictable situations. For example, due to situational changes, or changes in the environment.

Controlling dozens or thousands of robots is fundamentally different to controlling a unique entity, like a sheep-dog for example. Indeed, it is possible to teach the sheep-dog the meaning of *go straight*, *turn left* or *turn right*. The sheep-dog only needs to remember that the action *go straight* means that it does not have to change its direction, while *turn left* and *turn right* correspond to two different directions. What about now if we wanted to control a group of sheep-dogs? It is unlikely that, at each moment, they always face the same direction. Consequently, the commands *go straight*, *turn left* and *turn right* would no longer be meaningful, as they would mean different things to different sheep-dogs.

The same problem arise with swarms of autonomous robots. The same commands as above would not have the same meaning for each of them. Clearly, the operator cannot control separately each individual. This would not be scalable. He should not have to concern himself with each single robot. Instead, he should give

his orders to a group, expecting the group, as a single entity, reacts to these orders.

One of the important things we have to consider as designers of a control structure is the equilibrium between the level of control of the instructor, and the level of decision making of the robots. If the robots had a very high level of decision making, the human would not be able to control the swarm as precisely as he would like. On the other hand, if these robots would not take any decision by themselves, the instructor should control each individual, losing the benefit of swarm robotics efficiency.

3.2 Related Work

Interactions between users and swarms of robots have, to date, received little research attention. However, we can find in literature a few studies related to HSI. Most studies about swarm interaction are inspired from human robot interaction [2]. In this section, we review the related work in the field of HSI.

McLurkin *et al.* [47] propose some ideas to interact with an entire swarm. They developed a graphical interface which allows a single user to control a swarm of robots. They based their work on the idea of real-time strategy video games in which the players can manipulate an army of more than 100 units. The authors also discuss the problem of communicating information from robots to human. While the authors developed their solution with a graphical user interface, we think that a more natural way of communication is more suited. First, their solution requires the user more cognitive resources. He has to concentrate both on the graphical interface, and on the swarm he manipulates. In this work, using natural communications such as gestures tends to decrease the cognitive load of the user. Moreover, it removes devices dependency such as the mouse, the keyboard and even the screen. McLurkin's system needs real-time modelling of the robots and the environment. Our system allows the robot and the environment to be their own model [8].

Bruemmer *et al.* in [9, 15] propose that, instead of globally manipulating the swarm, the user could have some means of influencing the emergence of the swarm's behaviour. They point out that the interaction between humans and robots should not be inspired from the insect world, in which, according to the authors, the insects do not operate with human operators. The authors chose to make the human operator a member of the group in order to modify the behaviour. Our approach is fundamentally different, in that by allowing external manipulation of the swarm, we do not require a human operator to have to think like a member of a robot swarm.

Kira and Potter [34] use a different approach to let a human influence a swarm. It is based on *physicomimetics* in which forces that exist in the real world determine

the robots' movement. The authors study two interaction mechanisms. The first modifies the physicomimetics parameters in order to change the robots' behaviour. On the other hand, the second does not change these parameters but adds virtual agents in the system which interact with the real ones. In this work, we do not add any real forces in order to try to adapt robots behaviour.

In [36], Kolling *et al.* study the manipulation of large swarms of robots with simple commands to achieve basic tasks like rendezvous or deployment. They planned to study, in the near future, the efficiency difference in achieving specific tasks between a 100% autonomous swarm of robots and a swarm that can be helped by a human operator. In our work, we provide to the user more powerful tools than those given by Kolling. Moreover, like McLurkin, they developed a graphical interface which we believe, is not well adapted for real world communication control.

Recently, A. Giusti *et al.* in [23], developed a hand gesture system for communicating with a swarm. The real-time recognition of the hand gesture is processed by the robots themselves. They exploit robot mobility, swarm spatial distribution and multi-hop wireless communication in order to let the robots implement a distributed and cooperative sensing of hand gesture and reach consensus about a gesture.

3.3 High-Level Control

In this section, we present an abstraction of the main tasks that the robots must be able to perform in order to be useful for human beings. We will not focus on specific tasks such as *mine detection* or *object retrieval*. We suppose that the robots are already capable of doing several tasks like that. The problem that concerns us is how can a human operator order the robots to carry out a specific task in a specific location. We will see that this control must be equilibrated. The user must be able to control the swarm precisely while the robots must behave autonomously.

In figure 3.1 the black rectangles A and B represent two locations where a task must be executed. The gray rectangles model obstacles, like walls, that the robots cannot pass through. Finally, the set of blue circles depicts a single swarm of robots.

This picture models a place where the robots may have to work in different sites. Indeed, it is likely to happen that a human operator wants to request his robots to work on different tasks in different locations. These tasks can be of different natures thus many groups must work independently. From the picture 3.1, several difficulties related to the environment arise:

- The tasks to perform can be situated in many different locations in the environment.
- The environment is complex. There are several walls which create corridors

in which the robots must turn.
– It may happen that the passages are relatively small.

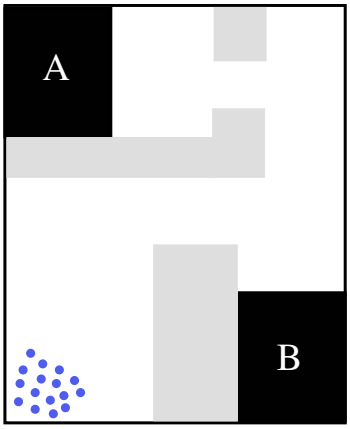


Figure 3.1 : Arena in which the human operator controls a swarm of robots. The black rectangles are two locations where the robots must go to work. The gray rectangles models the walls the robots can't pass through.

Moreover, the environment can change overtime. The robots must thus be able to move around an unknown environment.

Because the human operator knows the environment and locations of the tasks to perform, we decided to offer him the capability to guide the robots in this environment.

We saw that the environment in which the swarms had to move around was complex. The main task for the human operator is to move a swarm in the environment. To do so, a first solution would be that the robots know the exact position of where they must go and the path to reach this location. This is not a good solution because the system would not work in unknown environments. Hence, the robots should construct a map of the environment before being functional. However, any changes in the environment would lead to a desynchronisation between their virtual map and the reality.

Allowing the user the ability to steer a swarm in the environment provides the equilibrium between his level of control and the autonomy of each robot. The user only focuses on the direction of the swarm, not on the direction of each individual. It is up to each robot to autonomously take the right direction with respect to its own frame of reference. Moreover, it is the robots responsibility to stay close to each other in order to form a group at any time. The user does not have to worry about the structure of the group.

We also saw that the user must be able to send multiple groups to different places in order to work on tasks of various kinds. Therefore, we provide the operator with

a command to separate a swarm into two sub-groups. Once more, it is not the operator responsibility to choose the exact partition of the robots. Because the human only needs to manipulate groups, the user will not focus on anything but sending the split command to a group. The robots belonging to the swarm will autonomously separate in order to form distinguishable groups of approximatively the same size. Once the robots are separated, the user will be able to control the two groups independently.

On the other hand, an operator may want to add more robots in a group in order to improve the efficiency of a task. We let the user reassemble two groups in order to form a bigger one. Like in the previous commands, the human does not have to be concerned about the way the robots merge. After sending the merge command, the robots automatically decide where they will meet in order to form a unique group.

An interruption mechanism must also be provided to the human operator in order to stop a swarm either moving in the space or achieving a task.

Finally, all the previous commands could not be invoked without a mechanism for choosing the swarm which is intended to control. This is the selection mechanism which offers the user the abstraction of different and independent groups. This group abstraction is the key element for controlling multiple swarms, improving the effectiveness of the user.

3.4 Communication Architecture

In this section, we explain the architecture created to communicate the commands from the human control device to the robots.

3.4.1 Architecture

It was necessary to develop an architecture which allows the sending of action requests from a control device to the robots. We implemented a client - server architecture wherein robots act as clients and connect to a server from which they can receive the user's commands. This architecture is based on the Transmission Control Protocol (TCP) which provides a reliable connection between hosts [37]. The footbots used in this project are fitted with a 802.11g wireless controller by which they can be accessed.

As can be seen in figure 3.2, the control device software is coupled to a server which broadcast every known command. All the robots, which are connected to the server, receive a message that represents the command to be performed.

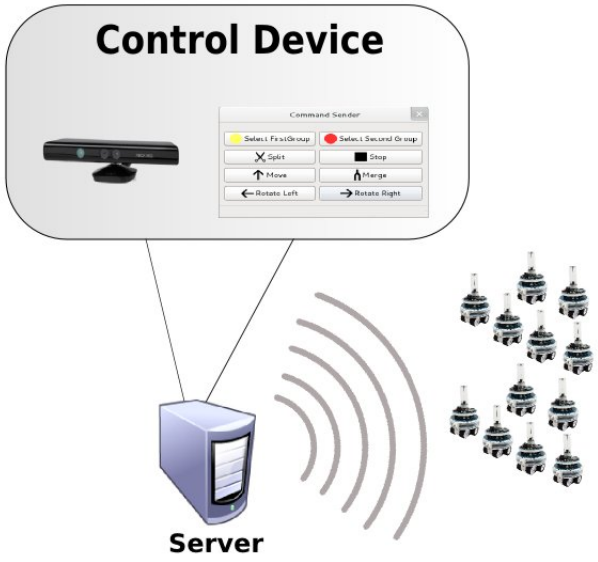


Figure 3.2 : The client - server architecture. The server broadcasts the messages to all the robots.

3.4.2 Protocol

To let the control device send commands to the robots, we had to design the communication protocol used to share the messages. In this section, we present the technology used to create the client - server architecture and the protocol that the server and the robots use to communicate.

We saw that ARGoS is coded with the Qt library. Consequently, it was tempting to use Qt and its QtNetwork module to easily and rapidly put in place a robust client - server mechanism. For our tests, this technology has been used with success. However, it introduces an issue. Indeed, real robots are ARM UNIX platform and Qt has never been successfully compiled on it, therefore, a pure UNIX TCP client-server had to be programmed. This has the advantages of being portable on every UNIX platform, i.e. the machine that runs the simulator and the real robots themselves.

We send a unique chararacter which contains a symbol that will be mapped to a command. This mapping is described in table 3.3

As seen in the table above there is a particular value, namely d, which represents the turn angle made by user's virtual steering wheel while he is turning the robots. This angle is sent in degree. When the robots' controllers receive a value that represents a number greater than 8, they either interpret it as an angle or discard it, depending on if the robots are moving or not.

Char	Command
1	Split
2	Merge
3	Move
4	Stop
5	Select First Group
6	Select Second Group
7	Rotate Left
8	Rotate Right
d	*Turn angle in degree*

Figure 3.3 : Mapping table from command symbol values sent by the server.

3.5 Low-level Controller

In this section, we present the controller's algorithm. This is the main function which receives every command and calls the corresponding behaviour. For more detailed explanations on the commands' algorithms, the reader may refer to section 3.6.

Although the robots run the same controller code, the action performed will be different for each of them depending on their inner state. Algorithm 1 shows an abstract view of the controller.

Algorithm 1: Controller – High Level View

```
1 begin
2     currentState = ϵ;
3     if New Command Arrived then
4         if I Am the Robot Who the Command Is For then
5             currentState = CommandState;
6     if My State matches currentState then
7         DoProperAction();
```

A state is associated to each robot and represents the action the robot is currently doing. There is a state for each command that lasts more than one control step. At each control step, the controller verifies if a new command has arrived and if it is concerned by that command it updates its inner state. Finally, the robots performs the action related to its state. It is clear that no new command arrived at each control step, but an action usually lasts more than one control step. Thus the robot always verifies if it is in a state that requires it to do something, and performs the action if necessary. Note that the empty state, represented by ϵ actually is an *idle* state in

which the robot does nothing but wait for a new command.

As seen in figure 3.2 the command sent by the server is broadcasted and each robot receives it. The robot must determine if the request is addressed to it. The system has been designed in such a way the user must select a group of robots before requesting it to do something. Two different selection commands are offered to the user: *select first group* and *select second group*. The first is used to manipulate one group. The user should always use this command before requiring them to do something. The *select second group* command is only needed to merge two groups. Consequently, when a user wants to send a command to a swarm, he first selects the group and then sends the desired action. Consequently, when the robots receive an action, they check if they were previously selected. If that is the case, then they start working.

3.6 Command Implementation

In section 3.3 we presented the tools available to the user in order to control swarms in complex environment. In this section, we describe the low-level implementation of the commands the user has at his disposal. We will start with the *selection* algorithm which is required before sending the other commands. Then, we will see the *move* which allows his to navigate a group of robots cohesively. We will see the *split*, which separates a group of robots in two subgroups. Afterwards, we will explain how the *merge* reassembles two sub-swarms in a single one. Finally, we will see why the *stop* is only needed for two commands.

3.6.1 Select

Selecting a group of robots is crucial in this system. The user must select a swarm before requiring it to do anything. The system has been designed in such a way the user cannot choose the group he wants to select. Due to this restriction, the mechanism must contain a way to automatically select a group, or change the current selection. To select another swarm, the same command is used: the user only repeats the command and the group selection changes. Moreover, we add the possibility to unselect the groups. The unselect command is actually performed when all the groups in the arena have been selected exactly once. For instance, if n swarms are in the arena, the $n + 1th$ selection command will unselect the currently selected group and will not select any other group.

The *merge* command is the only command which requires the user to select two groups at a time. A second selection command has thus been created, which works

as previously explained.

The following algorithm shows how the controller manages the selection.

Algorithm 2: Selection algorithm

```
1  FirstSelectedGroupID ← FirstSelectedGroupID + 1
2  if FirstSelectedGroupID == SecondSelectedGroupID then
3      FirstSelectedGroupID ← FirstSelectedGroupID + 1
4  while FirstSelectedGroupID < N and FreeGroupIDs[FirstSelectedGroupID] do
5      if FirstSelectedGroupID == SecondSelectedGroupID then
6          FirstSelectedGroupID ← FirstSelectedGroupID + 2
7      else
8          FirstSelectedGroupID ← FirstSelectedGroupID + 1
9  if FirstSelectedGroupID == N then
10     FirstSelectedGroupID ← -1
```

The intent of this algorithm is to update the variable that stores the current selected group. This variable is called `FirstSelectedGroupID`. At any time, its value must be the same in every controller. Indeed, each group must retain which group is selected, in order to determine if it is the next selected group.

The algorithm starts by increasing the first selected group variable, which should take the next existing group in the arena. However, this group may either not exist or cannot be selected for one of the following reasons:

1. There is only one group in the arena
2. There is a gap in the sequence of the group ids
3. The next group is already selected by the second selected command

The second reason may be due to a merge, which removes one of the two group numbers (see figure 3.4). In order to skip the gap, a vector of boolean variables is used to know if a group id corresponds to an actual group in the arena. This vector is called `FreeGroupIDs[]` and returns `true` when an id number does not match any existing group. Iterating over this vector allows the algorithm to reach the next existing group.

Remark that we limit the number of groups in the arena by a maximum of N groups. The reason is practical: a different colour is assigned to each group and this colour is chosen based on the group id. The controller has a fixed-length vector of predefined RGB colours and assigning it to a group is simply done by accessing the colours vector with the group id number.

The third reason why a group number could not be selected is that it is already selected with the second selection command.

false	true	true	true	true	true
0	1	2	3	4	5

(a)

false	false	true	true	true	true
0	1	2	3	4	5

(b)

false	false	false	true	true	true
0	1	2	3	4	5

(c)

false	true	false	true	true	true
0	1	2	3	4	5

(d)

Figure 3.4 : Evolution of `FreeGroupID[]` after a sequence of splits an merges. (a) is the beginning configuration with only one group. (b) a split has been done. (c) a second split occurred: three groups are in the arena. (d) group 0 and 1 have been merged and the system assigned the id 0 to the new group. A gap appears between group 0 and group 2.

The second selection algorithm is very similar to algorithm 2 but it first verifies if one group is already selected or not.

3.6.2 Move

The user which controls the swarm must be able to move it in the arena. He has to move the swarm as a unique entity, that is, he must have the feeling to move the entire swarm whatever its size, as he would lead a single robot. To understand how we move a swarm cohesively, we need to introduce flocking control theory.

Flocking Control

Nature abounds of groups of individuals that move in a cohesive fashion, that give the feeling that they are a unique organism moving and avoiding obstacles. This behaviour is called flocking and can be highly observed in birds or fishes. The first studies in artificial intelligence trying to create artificial flocking have been achieved by Reynolds [54] which realised the first simulation of motion of flocks. This work was based on three simple rules:

Separation: avoid crowding.

Alignment: move in the average direction of the flock.

Cohesion: adapt position to be placed at the same distance of neighbours.

We based our work on Turgut *et al.* [58] flocking control theory. The flocking control vector **f** may be expressed as:

$$\mathbf{f} = \alpha\mathbf{p} + \beta\mathbf{h}, \tag{3.1}$$

where $\mathbf{p}$ represents the force interaction between the robots, namely the proximal control vector and $\mathbf{h}$ is the alignment control vector.

Robots must be kept at a certain distance from each other. The proximal control vector models the attraction and the repulsion between the robots, so they can adapt their distance: when the distance with other robots is too high, they *feel* attracted to each other and move closer, while when the distance is too low, they *feel* repulsed and move away. This vector is calculated as:

$$\mathbf{p} = \sum_{i=1}^{k} p_i(d_i)e^{j\phi_i} \tag{3.2}$$

where d_i is the range, ϕ_i is the bearing of the i^{th} robot in the group and $p_i(d_i)$ is the magnitude of the vector. This vector length is computed by ($x = d_i$):

$$p_i(x) = |V(x)| = -\frac{dV}{dx} = \frac{2\epsilon\alpha}{x}\left[\left(\frac{\delta}{x}\right)^{2\alpha} - \left(\frac{\delta}{x}\right)^{\alpha}\right] \tag{3.3}$$

and where

$$V(x) = \epsilon\left[\left(\frac{\delta}{x}\right)^{2\alpha} - 2\left(\frac{\delta}{x}\right)^{\alpha}\right], \tag{3.4}$$

That is, $p_i(d_i)$ is obtained by calculating the derivative of a virtual potential force. It is said virtual because robots actually do not sense this potential force, but they have to compute it. In this work, we use a potential function which corresponds to the Lennard-Jones potential when $\alpha = 6$ [33]. δ is the desired distance between robots, and ϵ corresponds to the strength of the attraction and repulsion. The value of α used in our experiments is 0.25.

The alignment control vector, $\mathbf{h}$, is formally defined as:

$$\mathbf{h} = \frac{\sum_{i=0}^{k} e^{j\theta_i}}{\|\sum_{i=0}^{k} e^{j\theta_i}\|} \tag{3.5}$$

where $\|\cdot\|$ denotes the vector's length. Each robot has its own frame of reference, namely the *body-fixed frame of reference*. This reference frame is defined by the front of the robot ($x - axis$) and the $y - axis$ is coincident to the rotation axis of the wheels, it is right handed and fixed to the centre of the robot (see figure 3.5). Robots align by calculating the average orientation of their neighbours. In order for them to calculate

this, we need to add a common frame of reference. All the robots are outfitted with a light sensor and use a light source to compute their relative positions with respect to this light. A robot r computes its orientation θ_r, and receives the orientation θ_i of robot i with respect to the light source. After it has received the orientation of its neighbours, it calculates vector **h** by averaging their orientation.

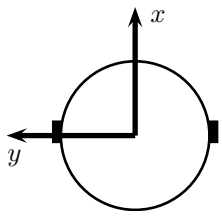


Figure 3.5 : Robot's fixed-body reference frame.

Robot Steering

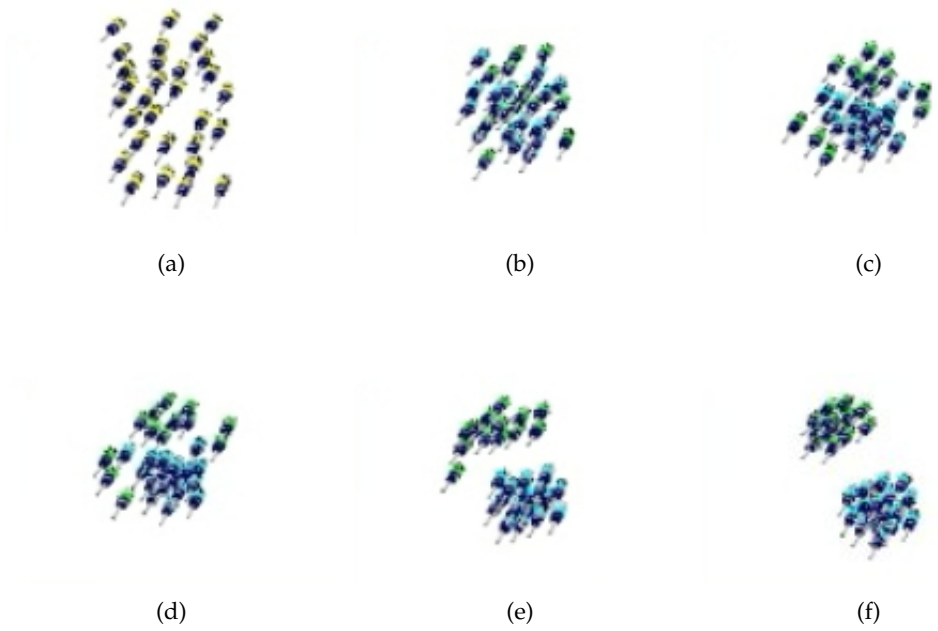
The user has the means to steer the swarm of robots in the arena. To do so, he just puts his hands in front of him, like he would use a virtual steering wheel. By turning this virtual steering wheel from left to right, he chooses the direction taken by the swarm he is controlling. The system reacts such that, the more he turns, the more the robots turn.

The steering mechanism has an implication in the flocking control of the group. When robots are moving, they try to stay close by the means of the flocking vector (see equation 3.1). Consequently, even if the swarm tends to move in the same direction, at each control step c, all the robots do not move exactly in the very same direction because they try to adapt their position with respect to the other. This means that, if we want an accurate steering mechanism, that is, robots turn in accordance to the angle made by user's steering wheel, we need to set the flocking vector **f** to zero as they begin to turn. This has the effect that the robots move forwards by turning independantly, that is, without flocking. As soon as the isuser puts his hands horizontally to make the robots advancing straight ahead, the proximal control vector **f** is set as previously explained.

3.6.3 Split

We now present the command which separates a swarm of robots into two subswarms. After selecting a group, the user may want to divide it so that he can move one group in one direction to do a task, and the other group in another direction, to perform a completely different task.

The idea of the algorithm is to randomly change the group id of approximately half of the robots and to assemble the robots of the same group in a specific place. Once the robots have chosen to either change their group id or not, robots of the same subgroup are spread among the other. Then, the two groups move in different directions. This is achieved by making the robots move cohesively: robots of the same group stay close to each other and are repulsed by the other group. Being repulsed and going in different direction clearly holds off the two subgroups (see Figure 3.6).



(a) (b) (c)

(d) (e) (f)

Figure 3.6 : Video sample from a split simulation.

We saw in section 3.6.2 that the Lennard-Jones potential is used to keep the robots at a stable distance from each other. There is an attraction force when the robots are too far and a repulsive force when the robots are too close. We use the same idea in this algorithm. The slight difference is that the repulsive force is bigger when the algorithm calculates it for two robots of different groups. This is achieved by setting the parameter δ at a higher value for robots of different groups than for robots of the same group.

Experiments shows that this method may already divide the initial group into some clusters which eventually join each others. However, we discovered empirically that we could accelerate the process by adding a direction vector, which is directed to the global centre of mass of the group in which robots belong to. This can be seen as an extra repulsion parameter, because the center of mass of the two

groups are not the same.

Algorithm 3 summarizes the split behaviour.

Algorithm 3: Divide a group into two subgroups.

```
1 SplittingGroups ← FirstSelectedGroup ∪ SecondSelectedGroup
2 if MyGroupID ∩ SplittingGroups ≠ ∅ then
3     c⃗ = CenterOfMass(MyGroupID)
4     v⃗ = FlockingVector()
5     d⃗ = v⃗ + αc⃗
6     SetWheelsFromVector(d⃗)
```

The parameter α allows us to raise the weight of the center of mass vector. In our experiments, this value is set to 4.

3.6.4 Merge

If the user thinks that a task may require more robots, he can merge two groups to form a bigger one. This section shows how we implemented the algorithm that reassembles two selected swarms (see Figure 3.7).

(a) (b) (c) (d)

Figure 3.7 : Video sample from a merge simulation.

The idea of the algorithm is to bring the two groups at half the distance that separates them. Arrived at this meeting point, one of the two groups changes its group number into the group number of the other group.

In order to know when the robots reached this distance, they calculate the time it takes to get there. By time, we mean the number of control steps that is needed to join the group halfway. This can be done because the robots move at a constant speed.

Each selected group moves in the direction of the other. This direction is actually given by the centre of mass of the future bigger group. Like in the previous algo-

rithms, robots move cohesively by calculating their flocking control vector. Once the robots started moving, the algorithm compares the number of remaining control steps until to reach the meeting point. Algorithm 4 resumes the operations performed by the controller.

Algorithm 4: Merge

```
1  flock = α Proximal + β GroupsCoM
2  SetWheelsSpeed(flock)
3  SetMergeData()
4  if MergeCpt ≥ ControlStepsNeeded then
5      StopMoving()
6      MyGroupNumber ← FirstGroupMergingNumber
7      FreeGroup(SecondGroupMergingNumber)
8      MergeCpt ← 0
9      SetIdleState()
10 else
11     MergeCpt ← MergeCpt + 1
```

The merge algorithm 4 is called at each control step by the two groups that are merging. It moves the robots in the direction of the other group. After the merge counter (number of control steps elapsed since they start moving) reached the total number of control steps needed to move to the meeting point, it stops the group, updates its group number and releases the group number of the second group.

Note that the groups that are not merging also need to know that two merging groups have joined so they can update their own list of free group id. This is done by catching the merge command, even if it is not addressed to them.

3.6.5 Stop

The only commands which the user needs to explicitly stop are the move and the split. Because the split algorithm does not have any means of knowing that the two groups are actually clearly separated, the user needs to tell it to stop. In both cases, stopping a group from doing something only stops their wheels and changes the controller's state to *idle* mode.

3.7 Real Robot Experiments

Each algorithm presented in section 3.6 were executed on the real robots. We have been able to reproduce in real conditions the set of actions we were able to do with the simulator. These actions were requested by the means of the same gesture recognition system developed as part of this thesis (see Chapter 4).

Even if the result of each behaviour were reached, we encountered some issues with the split behaviour and the move. During a split, some robots did not respect the attraction and repulsion force. Indeed, it has happened that two robots moved in such a way that they both went in the direction of the other to eventually bump. By decreasing the velocity of the wheels and by increasing the minimum distance between the robots, this problem was solved.

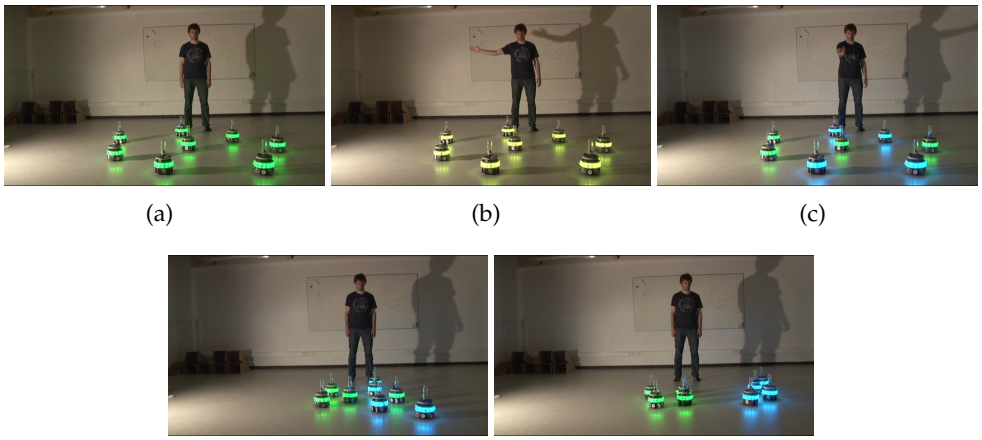
A second difference with the simulation is that the steering mechanism is less easy to use. We saw that the robots had to stop flocking when they start turning. After having turned, the robots communicate in order to flock. This introduce at this moment, a small bias in their direction. Due to the flocking mechanism, the robots do not continue in the very same direction and deviate.

Despite these small issues, we were able to send every command to the robots. They performed the behaviours as we expected, in spite of some small differences with the simulation.

Figure 3.8 illustrates the split behaviour. First, the user selects the group he wants to split. Then, he executes the gesture associated to this command. The robots change their group number with probability 0.5. They feels attracted to the robots from the same group and repulsed from the robots of the other group. Figure 3.10 shows two groups merging. First, the user selects the groups. Then, he orders them to reassemble in a bigger one. The two groups move in the direction of the other. Arrived mid-way, the two groups are reassembled. Figure 3.9 depicts a user guiding a swarm. He steers the robots with its arms like if he has a steering wheel.

3.8 Discussion and Conclusions

The work presented in this chapter contributed to the swarm robotics part of the interaction between human and robots. First we designed a network architecture in order to send data from the control device to the swarms of robots. The resulting architecture broadcast the gesture command label. Based on this command and on their inner state, the robots decide if they must perform the action or discard the command.



(a) (b) (c)

(d) (e)

Figure 3.8 : Split: (a) initial group (b) group is selected (c) the user orders the robots to split (d) robots of the same group are attracted. (e) the two groups are separated.

(a) (b) (c)

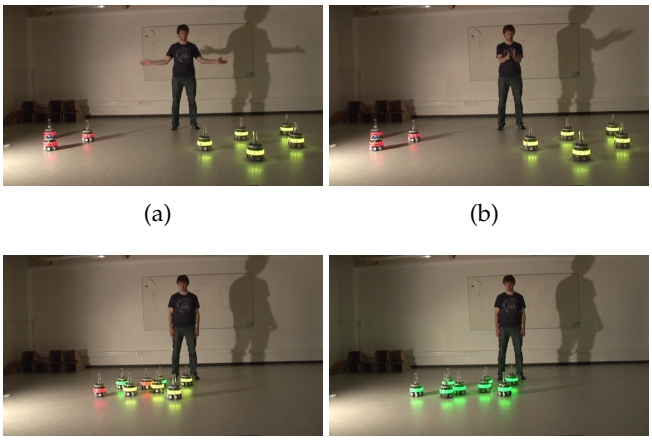
Figure 3.9 : Move: (a) the user steers the robots with its arms. When its arms are are the same high, robots move straight. (b) the user turns its arm on its right to turn the robots. (c) finally the robots continue straight, according the user order.

We also proposed the minimum useful swarm behaviours:

Move is the action consisting in steering the robots in the arena. The Kinect system offers the user a virtual steering wheel with which the user chooses the direction of the swam while it is moving.

Split separates a swarm of robots in two sub-groups. The user does not do anything but send the split command and the robots separate by themselves according to attraction and repulsion rules. A weak point of the current algorithm is that the robots receive very precise range and bearing information of all the robots involved in the split. This is not the case with the real robots because of noise and range and bearing limitations. Robots should communicate only with a restricted neighbourhood. Each robot could propagate data from other robots by hopping from robot to robot.

Merge is the opposite command of the splits. It reassembles two swarms in a single



(a) (b)

(c) (d)

Figure 3.10 : Merge: (a) two groups are selected (b) the user orders the groups to merge (c) each group go in the direction of the other one (d) at mid-way they finished each robot belong to the same group.

one. The swarms can be anywhere in the arena but must be face to face. An interesting improvement would be to allow a user to merge two groups of robots even if they are separated by a wall. The groups should find a path in which they eventually meet.

Select is available in two commands. One is used to take the control of one group at a time. A second command exists to achieve a merge.

Stop the robots from moving.

All these behaviours were created in order to have an equilibrium between the user precision control and the auntonomy of the robots.

Chapter 4

Gesture Recognition

This part of the work was carried out jointly with Youssef S. G. Nashed, PhD student at University of Parma, Italy. For his internship, Y. S.G. Nashed had to develop a gesture recognition software. We decided to work together in order to achieve our respective goal.

Y. Nashed planned to develop new gesture recognition techniques during his PhD. The first version of the system developed at ULB for his internship, is used in this work.

This chapter is organized as follows. First, we introduce the hardware device, the imaging device that retrieves the user's body data. Then, we will motivate our choice regarding the framework used to build the system. Subsequently, we will present the methods and algorithms developed to have a usable gesture recognition system.

4.1 Hardware

This section presents the Microsoft Kinect Device, a real-time 3D device initially created for the Xbox video game console.

Kinect is embedded with an RGB coloured camera, a depth sensor and a multiarray microphone. The depth sensor is divided in a IR light source and a monochrome CMOS sensor in order to calculate the distance between the sensor and the objects. Thanks to the infrared, the device can be used in any ambient conditions.

4.2 Software

This section presents the different frameworks that allow us to communicate with the Kinect device. Since the Kinect is quite a young device, we only focus on the framework that was available at the beginning of this work.

4.2.1 Microsoft Kinect SDK

Microsoft waited eight months before releasing their software development toolkit. Microsoft announced the availability of their SDK for non-commercial use [1] in June 2011 [1].

The API provides a data structure which allows the programmers to manipulate the user's skeleton. As shown in figure 4.1, the structure offers the developers information about 20 joints of the user's body.

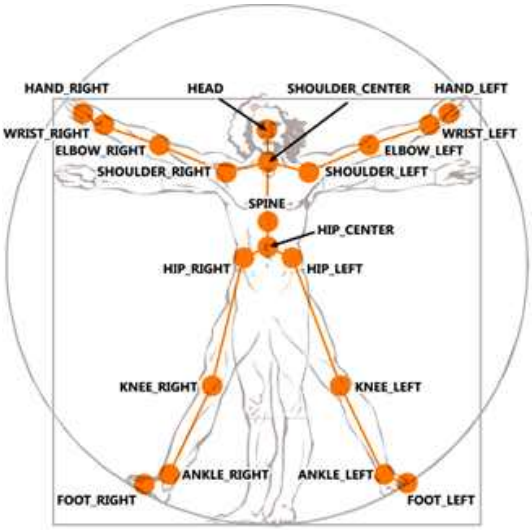


Figure 4.1 : The 20 joints that Microsoft Kinect SDK recognizes. (Source: `http://msdn.microsoft.com`)

Unfortunately, the only information about the joints are their position in the space. We will see later in section 4.3 that joints positions are inconvenient data to achieve gesture recognition.

A second disadvantage of this framework is that it only runs under the Windows 7 operating system. One of our requirements was to use a cross-platform API in order to develop a system that can run under, at least, both Windows and Linux.

1. As of February 2012, Microsoft released a new version of its SDK which is now commercial ready.

4.2.2 OpenKinect

While the device was released by Microsoft on November the 4th 2010, the first open-source [2] driver was commited on GitHub, a web developers platform [3] on November 10th, 2010.

OpenKinect provides a driver and an low-level API to communicate with the device under Linux, Windows and Mac. This means that it only provides a mechanism to access raw data such as RGB and depth images, controlling the motors, the accelerometers and the LEDs.

Compared to the Microsoft SDK, the disadvantage is that no higher-level functionalities are available for the programmer to manipulate user's skeleton data. In order to develop a gesture recognizer, more complex work on raw data should be performed to achieve the same result than a framework providing higher data structures.

4.2.3 OpenNI/NITE

PrimeSense [4] is the company that developed the motion sensing technology of the Kinect device. PrimeSense created OpenNI, a non-profit organization which aims to provide a set of API for writing NUI applications for several devices such as the Kinect.

The OpenNI organization developed a framework, confusedly called OpenNI, which provides an interface for physical devices like the Kinect. This framework is free and open-source [5]. OpenNI can be coupled with NITE, a middleware providing motion tracking which is also distributed free. On the other hand, the source code is not open. NITE provides the implementation for manipulating the user's skeleton data such as the joints position and orientation (see figure 4.2).

Figure 4.3 depicts the architecture of OpenNI. NITE corresponds to one of the middleware components and implements the skeleton data structure. The application uses this structure through the OpenNI interface.

For this work, this framework was chosen because it corresponds to the requirements of the system that is developed. It is mutli-platform and it provides sufficient high-level functionalities to implement a simple gesture recognizer.

2. The licences used by OpenKinect are Apache2.0 and GPL2.0.
3. `https://github.com/OpenKinect/libfreenect`
4. `http://www.primesense.com`
5. GPLv3.0 and LGPLv3.0

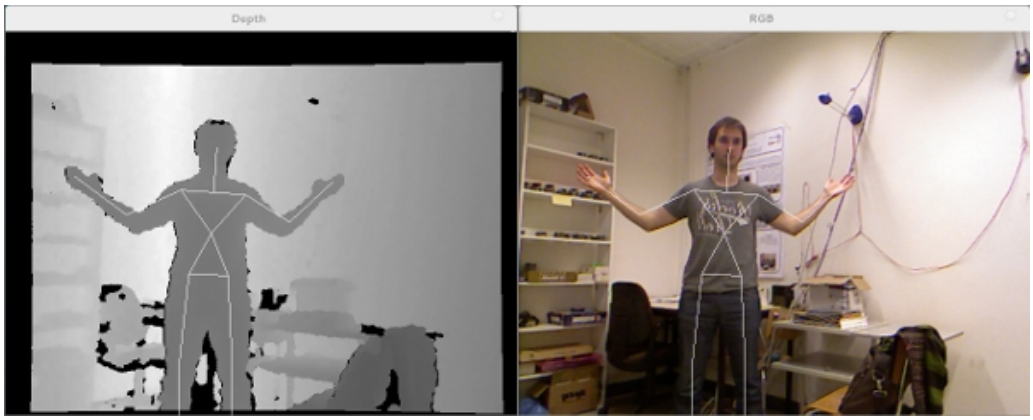


Figure 4.2 : On the left, the depth image with the skeleton tracking. On the right, the RGB version and the skeleton tracking.

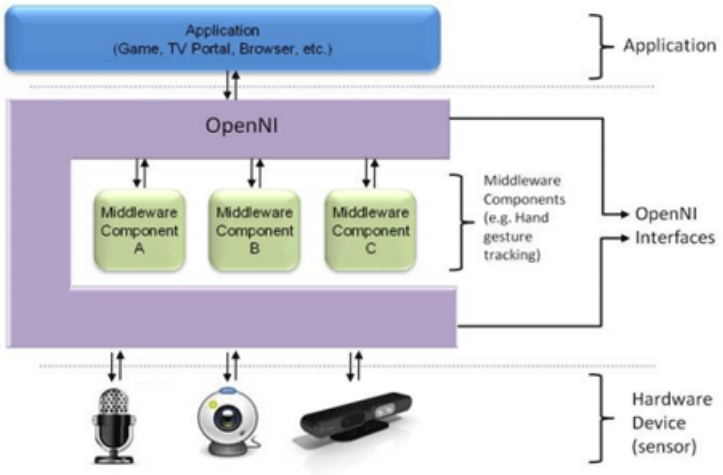


Figure 4.3 : OpenNI/NITE Architecture

4.3 Gesture Processing

The human body is structured by its skeleton composed of 206 bones [53]. These bones are connected by articulated joints permitting movements. All joints do not have the same number of movement direction: they have various *degrees of freedom* (DOF). For instance, the knee is a 2-DOF joint: flexion, extension and lateral rotation, while the shoulder is a 3-DOF joint. We will see in the following how we use joints to define and recognize gestures.

In the previous section, we have reviewed the three main frameworks which can control the Kinect device. The framework chosen to develop the gesture processing is OpenNI jointly with the NITE middleware. In this section, we will see the algo-

rithmic aspects of the system. First, we will give a high-level introduction of the skeleton structure provided by OpenNI. Then, we will discuss how we created the gestures. Subsequently, we will show how these gestures are recognized.

4.3.1 The Skeleton Data Structure

The API provides positions and orientations of the skeleton's joints of a user with respect to the real world coordinate system, where the origin is at the Kinect sensor (see figure 4.4).

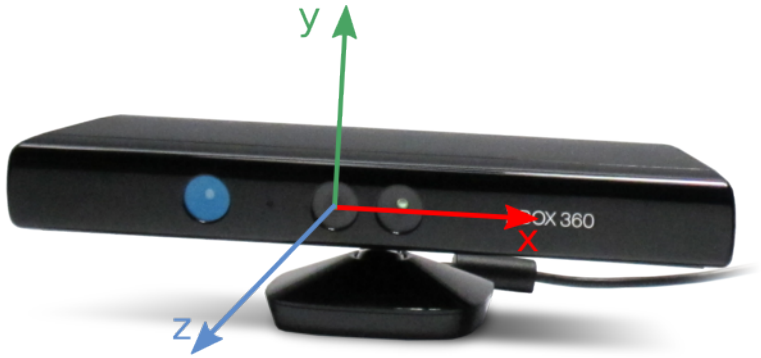


Figure 4.4 : The real world coordinate system takes its origin at the device sensor with the $z - axis$ pointing out of the screen.

In the following, we present the differences between the use of joints positions or the use of the orientations in the perspective of implementing a gesture recognition software.

Position

The API returns the position of a specific joint as a 3-dimensional vector containing the X, Y and Z values with respect to the real world reference frame. These coordinates are given in millimetres

It is clear that, given two different locations of the user in the space, the API will return different values for his joints' positions. Moreover, each pair of joints are separated by a distance which is different for nearly everybody. Thus, for two different people in the very same position, at the very same place in the space, the values returned by the API have a very high chance to be different

We can see that using the joints' position to develop a gesture recognizer is not a good idea. It would be too user-dependant and the user would always need to

be placed at the same place in the environment, which is not always possible if this environment changes.

Orientation

In addition to the joints' position, the framework also returns the orientations of the joints as a 3x3 orthonormal rotation matrix. It represents the rotation between the joint's local axes and the real world coordinates. The API defines the *T-pose*, which is the neutral pose in which each joint's orientation is aligned with the real world coordinate system (see figure 4.5).

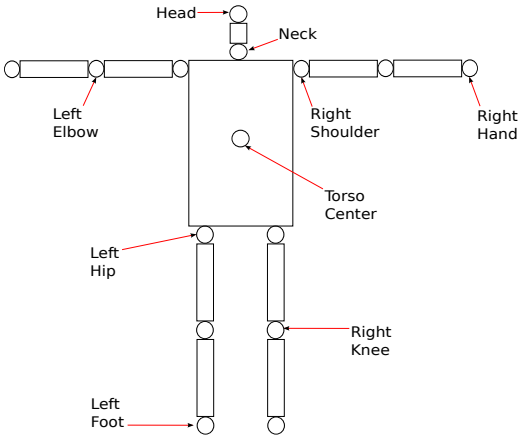
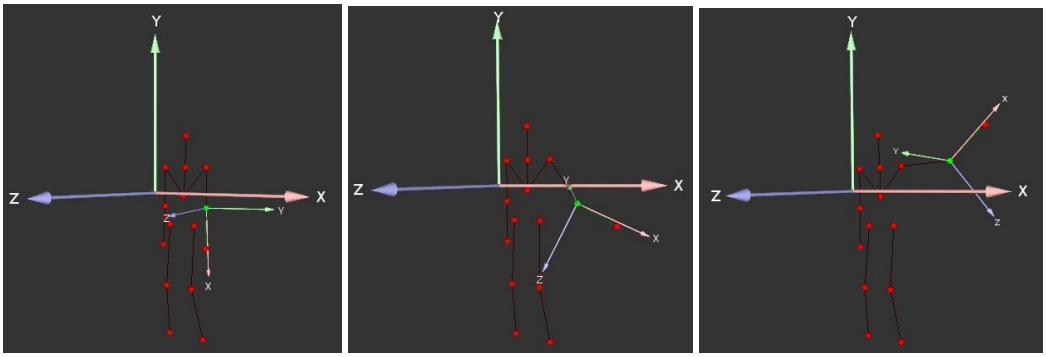


Figure 4.5 : Skeleton representation. Source: based on *Joint definitions* from [52].

The following figure depicts the evolution of the user's right elbow rotation with respect to the world coordinates.



(a) (b) (c)

Figure 4.6 : Joints orientation: the big axes are the world coordinates system. The small ones are those attached to the users' right elbow.

If the rotation matrix is used to manipulate the skeleton's joints, the disadvantages of using joints' position mentioned above disappear. There is no more dependency on a specific user's skeleton: if two different users are in the same position, the joints' rotation must be the same for each of them.

4.3.2 Gesture Creation

Creating a gesture informally consists of taking "snapshots" of several poses of the user's skeleton. From these snapshots, information about the skeleton must be saved. We saw in the previous section that the OpenNI framework can access joints information such as position and orientation. Unfortunately, the position of the joints in the space is too user-dependant. On the other hand, the joints' orientation can be used to determine a pose without requiring any information about the place in the space where the user is.

This is the approach that has been implemented in the system developed as part of this thesis. Each gesture is defined by a finite ordered sequence of poses. These poses are characterized by the joints' orientation. These orientations, expressed as a rotation matrix are saved for each joint involved in the gesture. Technically, for each pose, one XML file is created in which joints' rotation matrix is formatted in XML tags.

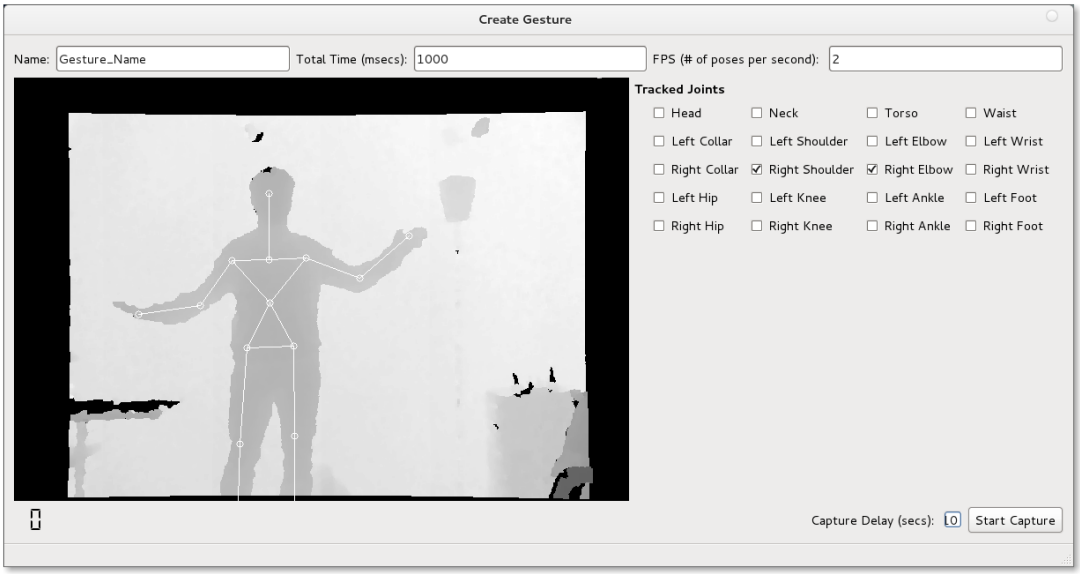


Figure 4.7 : The graphical interface for recording a new gesture.

Figure 4.7 depicts the graphical interface developed to record users gestures. On the left of the interface, the user's body is shown with its skeleton. Before recording

the gesture, it is required to select all the joints that are involved in the gesture. Then, the total time of the gesture should be entered (generally one second) and the number of poses per second that the user will record. Afterwards, each pose is saved by the system by taking the relevant data from the skeleton.

4.3.3 Gesture Recognition

Gestures are a natural way to the communicate. Whether it is by finger, hand, arm, face, head, a lot of meaningful information may be expressed by simple movements of the body. If humans are used to recognize the meaning of several gestures even without any previous explanation about them, computers must know the difference between a meaningful gesture and a body movement that is of no interest.

Mitra and Acharya [48] broadly define gesture recognition as being the process by which the gestures made by the user are recognized by the receiver. A gesture can be either static or dynamic. A static gesture denominates the set of poses performed by a user, while dynamic gesture is used to categorize the gestures made by a set of motions in a single continuous flow.

There exists several approaches in the literature to achieve gesture recognition. For static gestures, a template-matcher may suffice, while for dynamic gestures, structure with temporal dimensions should be used. Lee and Kim propose in [39] an approach using Hidden Markov Models. Israd and Blake introduced a condensation technique, which is an improvement of particle filtering for tracking process [27]. Davis and Shah developed in [16] a method to recognize human gestures using a Finite State Machine (FSM). The method used in this work to recognize user's gesture is based on a FSM.

Gesture Abstraction

In order to create a gesture recognizer, the body motions must be abstracted and modelled. There are two main categories: appearance methods and 3D model methods. The first one uses methods such as body marker, or geometric parameters of the body (perimeter, convexity, . . .). The second method works with skin information, skeleton and joints data.

The OpenNI/NITE framework is based on a 3D model. It provides different data about the 3D spatial information of the user's joints.

Algorithm

The idea of the algorithm is to assign a deterministic finite state machine (DFSM) to each gesture that must be recognized. Matching a gesture is achieved by reaching the (unique) final state of its DFSM.

Each state of the DFSM corresponds to a pose of the gesture it models (see figure 4.8). The final state is the last gesture's pose. The transitions are done by evaluating the next pose in the ordered sequence of the gesture. Moreover, a timer is assigned to each state and is launched when the DFSM is in that state. The value of this timer is set to the total time of the gesture divided by the number of poses. If a timeout occurs, the DFSM goes back in the start state.

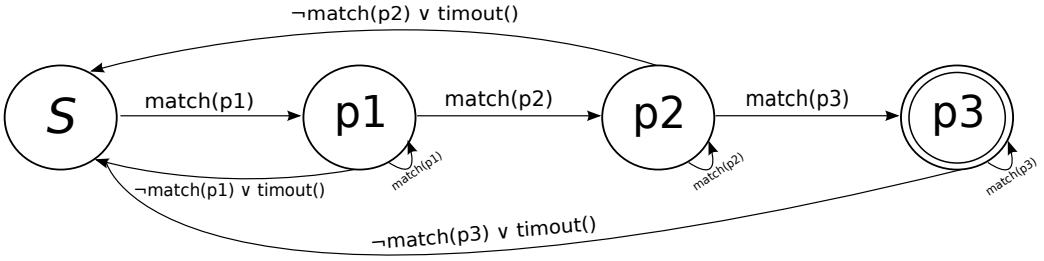


Figure 4.8 : Deterministic Finite State Machine assigned to a gesture: it goes to the next state if the kinect current pose match as the next pose in the ordered sequence of the gesture.

Several gestures must be recognized, consequently, there are as many DFSM as gestures. At each frame of the camera, the skeleton information is returned by the framework and a pose object is created. This "frame pose" is sent to each DFSM as input and they analyse it with respect to their state.

Matching a pose is done by comparing, for each Kinect frame, the orientation of the skeleton joints to the orientation of the joints recorded. This is achieved by doing a dot product between the X, Y, Z rotations matrix's vector. If the orientations were exactly the same, the dot product of the rotation matrix vector should be equal to 1. Because it is very unlikely for the user to be in the very same pose (that is, with the same joints' orientation) the dot product is compared to an *accuracy coefficient*. Therefore, the pose is a match if the dot product is higher than this coefficient. During our experiments, its value was set to 0.8. This means that the users' joints orientation can diverge from a maximum of 20% (in the X, Y, Z directions) of the position they

recorded (see Algortihm 5).

Algorithm 5: Pose Match

```
CurrentFramePose ← GetSkeletonPose()
foreach Active CurrentFramePose's joints (numbered j) do
    X1 ← X-axis of the jth joint of the current frame skeleton
    X2 ← X-axis of the jth joint of the DFSM state pose
    Y1 ← Y-axis of the jth joint of the current frame skeleton
    Y2 ← Y-axis of the jth joint of the DFSM state pose
    Z1 ← Z-axis of the jth joint of the current frame skeleton
    Z2 ← Z-axis of the jth joint of the DFSM state pose
    if (X1 × X2) < ACCURACY_COEFFICIENT then return False
    else if (Y1 × Y2) < ACCURACY_COEFFICIENT then return False
    else if (Z1 × Z2) < ACCURACY_COEFFICIENT then return False
    else return True
```

4.4 The Set of Gestures

We defined the set of gestures the user is able to do in order to control the swarms. The reason why we fixed the set of gestures is that the gesture recognition algorithm is less accurate with some joints orientation [6]. Thus, we empirically found a set of relatively intuitive gestures that are correctly recognized by the system. Figure 4.9 depicts the motions of each gesture.

4.5 Discussion and Conclusions

The work done in this chapter was performed with the collaboration of a PhD student from the University of Parma, Italy. A minimalist gesture recognizer has been created with the help of the multiplatform framework OpenNI.

As we have seen, the algorithm uses a deterministic finite state machine for each gesture which has to be recognized. The DFSM goes from state to state each time the algorithm matches a pose belonging to a gesture. The matching pose is done by comparing the orientation of the body joints. Even if the joints' orientation should be body-independant, we noticed during our tests that depending on the skeleton, some gestures are more difficult to recognize. Consequently, we decided to fix the set of gestures the user can perform and to record the gestures of each user before

6. This is essentially due to the framework which does not provide their orientation with a high level of confidence

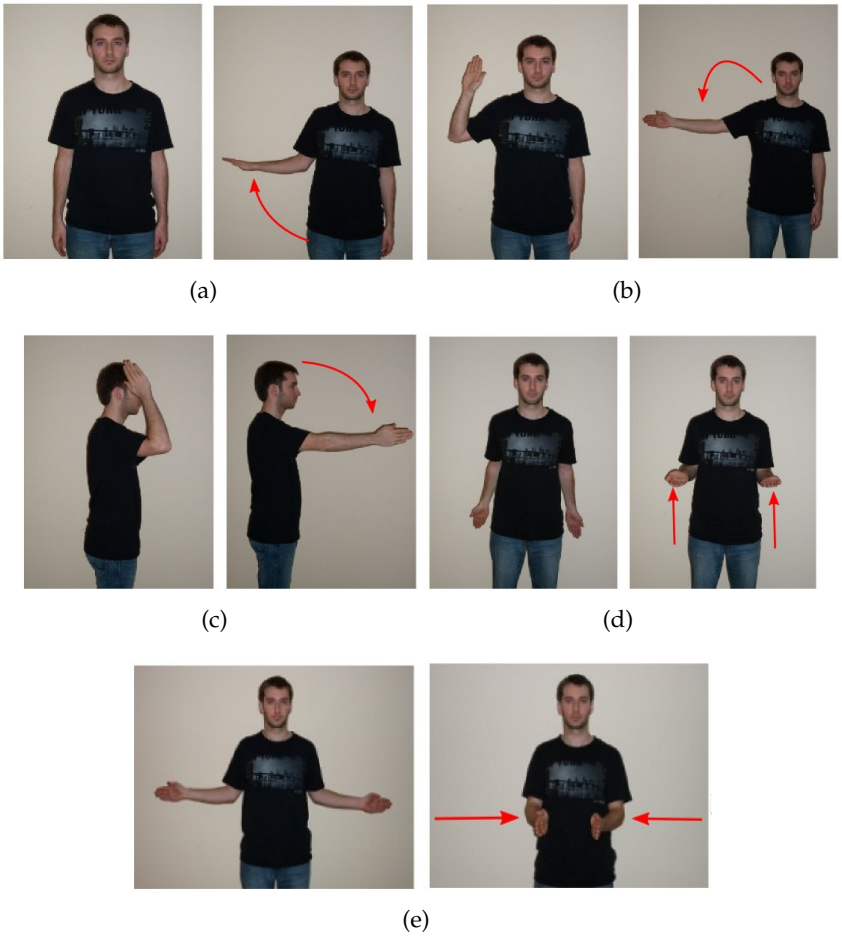


Figure 4.9 : (a) Stop. Slide up the right arm to form an angle of 90 degrees with the body. (b) Select. Slide down the arm horizontally. (c) Split. Down the arm in front of oneself, like if one cut something in the middle. (d) Move. Move up both arms in front of oneself. (e) Merge. Bring the arms towards each other.

starting to use the system. For a more accurate system, it could be interesting in the future to implement other kind of techniques, as seen in section 4.3.3.

Chapter 5

Usability Experiments

We have developed a way to communicate with swarms of robots. This system, based on the Kinect device, uses human gestures in order to interact with the robots. Is this new interaction system usable? Can we, effectively manipulate swarms of robots and can we really do it with a natural interaction device?

The experiments in this chapter show that a user is effectively able to take the control of a large number of robots. To do so, we put in place usability experiments. These experiments are useful to study the ease of the manipulation of a system or to highlight potential drawbacks in its design.

This chapter is organized as follows. First, we will see the importance of measuring usability. Then, we will present the proposed methodology to study the system's usability and we will see some results of this methodology applied to the robots control system.

5.1 Motivation

What exactly is usability? Can usability be rigorously measured? These are probably the two most frequently asked questions when we talk about usability of a system. No one would use a system daily that is too complex to understand, that is confusing or that would take too much time to perform a simple task. Worse than not using a system, Nielsen [29] cites a study in which 22 usability problems in a medical institution cause the patients to receive the wrong treatment. Usability is thus a high concern in developing interaction systems.

Creating a correct and functional design without any users feedback is almost impossible. Different aspects of humans play an important role in the development of a new interaction system. Educational, cultural or sociological dimensions are

factors that modify the perception and the usability of a system [21].

Non-formally speaking, Hartson [26] defines usability as the way to say that a design is "good" in terms of human interaction. To make things clearer, we can focus on the International Standards Organization usability definition (ISO 9241-11): *"Extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"*. Three keywords may be highlighted:

Effectiveness is the accuracy with which the user achieves a specific goal.

Efficiency is the resources used to achieve the goal. For example, the time taken to perform a specific task.

Satisfaction is a subjective evaluation regarding the comfort of the user. Usually, a satisfaction questionnaire is filled during or after the experiment.

As we can see, usability is not a one-dimensional property of the system's use. An aggregation of these three concepts may give a strong idea of the usability of the system.

5.2 Experiments

Our system uses gesture as the means for controlling the robots. A user does gestures that are linked to commands that the robots perform. We thought that this interaction would be very easy to use and very intuitive for any kind of people. In order to verify that our thoughts were true, we had to give a measure to the users' experience of this system.

In order to be able to take this measure, we developed a very specific scenario. This scenario consists in a set of tasks the users must perform with the robots. This allows us to take different kinds of measures. Because the same scenario were executed by every user, these measures could be compared between the subjects.

This section is divided in two subsections. First, we present our methodology, that is, the scenario performed by our subjects, the number of participants, the categories of users and the measures that are studied. Then, we will show that the Kinect system is capable of providing the means to take the control of a potential huge number of robots at a time.

5.2.1 Methodology

Humans play the central role in the experimental process and each experiment must be executed the same way if we want to be able to compare the data.

In order to plan a good experimental process, we need to determine which metrics we are going to collect. However, it may be hard to know what are the metrics in a usability test. We manipulate different kind of metrics daily. We can easily measure the size of a person, the temperature of water, the speed of a vehicle and so on. Usability metrics may not be as obvious, but they exist. Indeed, we can measure, among others, the amount of task success, the time taken to perform a task, the amount of errors and the user's overall satisfaction.

Usability metrics must give useful information about the user's experience and the interaction between the user and the system. This information must be observable and quantifiable. All the data we have collected were numbers, even the global user's satisfaction based on a questionnaire will be transformed into numbers.

In order to know which metrics we will need, we must determine what is the final goal of our experiments. There are two kinds of usability measures [5]:

Formative is used to identify usability issues in the system. The goal is thus to improve the system during its development phase. Usually, an iteration process for testing is established. After each usability test, the developers fix the problems revealed by the experiments.

Summative corresponds to a global evaluation of the system. It may be used to make comparison between products and to determine whether the product achieves its usability goals. It is usually evaluated by evaluation score, task-time or completion rates.

In our case, what we want to measure is the ease of the manipulation of the system. That is, to what extent is gesture control relevant to manipulate swarms of robots. We have decided as part of this work to evaluate the final product developed during the academic year. Summative measures are consequently more relevent for us. It will give us an overall evaluation of our system.

In order to create our summative evaluation process, we developed another means for controlling swarms of robots. The idea is to compare our new interaction system to one that users are more accustomed to. The result is a very simple graphical user interface (see figure 5.1) that contains a button for each available command. This graphical interface is used through a mouse device or a keyboard device. This will help us with comparing the ease of the manipulation between a new interact device and a very common one.

Subsequently, we will present what users had to do during the experiments. Then, we will see what kind of users we decided to test and how many participants we used in this usability experiment. Afterwards, we will talk about the actual data we have collected from the experiments, that is, the time on task data and the user's satisfaction.

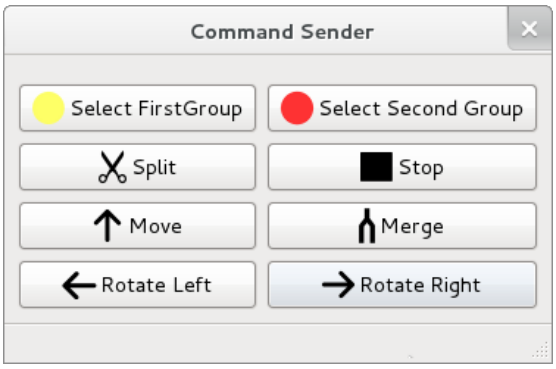


Figure 5.1 : Graphical user interface containing a button for each command. Clicking a button sends the command to the robots.

Scenario

The scenario performed by the users determines the data we can collect. Indeed, the more different actions the user does, the more data is available. Consequently, the scenario must contain enough elements to get useful information about how the user manipulates the system.

The scenario that we developed requires the subjects to do a fixed set of actions that use at least each command once (see figure 5.2). Initially, the system starts with a unique swarm of robots. The goal of the users is to create three sub-groups and to move them in a specific zone of the arena, namely the *task zones*. A task zone is a delimited area in which the robots simulate working. The arena contains three task zones, one for each group of robots. Once each swarm has been in a task zone, the subjects must reassemble the three groups in a unique swarm anywhere in the arena.

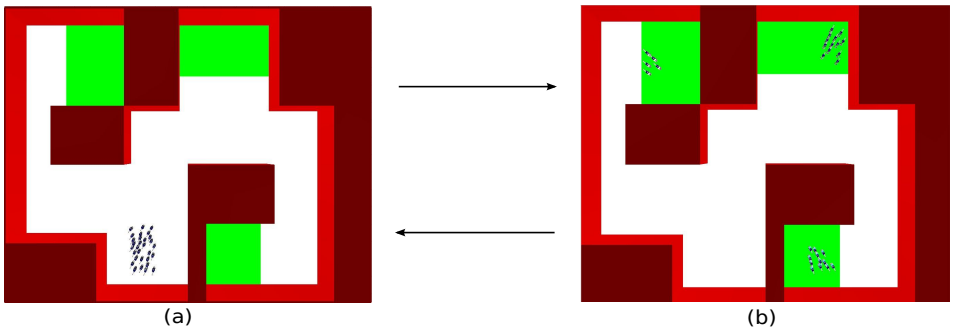


Figure 5.2 : (a) The arena that contains the initial group of thirty robots. (b) Each group of robots is moved in a different task zone, modeled by the green rectangles.

Sample Size

Literature about usability test sample size abounds of debate regarding the better number of subjects for an experiment [45]. Formative evaluation focuses on problems' discovery. A popular but criticized [60] article from Nielsen [28] argues that five participants will discover 80% of the problems in a system. In any case, a small amount of users, that is, generally less than 10 subjects is sufficient in a formative usability evaluation [50].

However, in a summative usability test, there is no consensus about the sample size. From a statistical point of view, the more data is available, the more precise the results. As part of this work, we were able to conduct 18 experiments.

Time on Task

When we want to compare the efficiency between two systems, a natural measure is the time it takes to perform a certain amount of tasks. In most systems, the quicker is the better. The user generally does not want that a task requires him too much time.

For our experiments, four main tasks can be extracted from the scenario. Recall that the scenario requires the user to divide one group in three subgroups, to move them in a different task zone, and finally to reassemble the three groups in one group. Based on these actions, we will measure the time taken by the user to move each subgroup in a task zone. That is, each time a group enters a task zone, we save the time taken to reach this zone. Because there are three groups for three task zones, three time measures are collected. Finally, we measure the overall time taken by the user to achieve the goal entirely. Because parallel control is possible, this last measure clearly can not be calculated by the sum of the three previous tasks plus the time it takes to merge the three groups.

We must define the beginning and the end of a task. We decided to launch the counter when the first group is selected. When a group enters a task zone, they automatically start blinking to communicate the fact that they are working. Once they start blinking the watch is stopped and the value is saved. For the overall time of the experiment, the start time also is the moment when the initial group is selected, and the final time is taken when the two last groups finish merging.

What we actually measure is the number of control steps. This number is linked to the time. In the configuration file of the simulation, one can define the number of ticks per second, that is, the number of control steps per second. In our experiments, we set this value to 10.

Usability Questionnaire

We saw that time-on-task measures can be useful for collecting data about efficiency of the system. However, this kind of data does not give any information about the overall satisfaction of a user. A system which is funnier to use may be more appreciated than a system which is repetitive or without any attractiveness but which takes less time. Consequently, it may be useful to study the overall user's satisfaction.

A good way to gather such results is for the subjects to complete a questionnaire. There are two moments to collect these data: after each task and at the end of the experiment's session. The first uses a so-called post-task rating questionnaire while the second uses a post-study rating questionnaire [56]. Post-task questionnaires are very short, varying from one question (SMEQ, UME) [46, 61] to three questions (ASQ) [40]. Post-session questionnaires themselves contain more questions and do not focus on any particular task.

A post-task questionnaire may be very useful to get data about specific tasks in the system. For example, if the aim of the experiment is to test an e-mail client, one can ask the user to give his feeling about the specific task "sending an e-mail to contacts registered in the addressbook" or "downloading the attachment of an email". However, the aim of our system is to interact with swarms of robots and what we want to measure is the ease with which the user controls the robots. A post-task questionnaire is consequently not appropriate and we preferred the post-session questionnaire which gives a general judgement of the system.

A satisfaction questionnaire is composed by a certain amount of items, which can be open-ended questions or can look like Likert-scale affirmations. In a Likert-scale evaluation, the statements are accompanied by either a 5-scale or 7-scale from "strongly disagree" to "strongly agree".

So far, we saw that using a post-session satisfaction questionnaire may be useful to study the users' global feeling of the system. Even if one may think about some relevant questions, standardized questionnaires are preferentially used. The advantages of standardized questionnaires are their psychometric validity and reliability [49]. The validity of a questionnaire consists in verifying that the questions are directly related with what the questionnaire is supposed to measure. Reliability is the consistency of measurement, that is, the fidelity of the score with respect to the questionnaire. It is evaluated with a coefficient alpha, namely the Cronbach's coefficient and varies from 0 (non reliable) to 1 (totally reliable) [13].

In [30], the authors report the four most used standardized post-session questionnaires:

– Questionnaire for User Interaction Satisfaction (QUIS) [12]

– Software Usability Measurement Inventory (SUMI) [35]
– Post-Study System Usability Questionnaire (PSSUQ) [43]
– System Usability Scale (SUS) [7]

Each questionnaire has its advantages and disadvantages. However, the two first one require a non-free license [30] so we decided to focus on the PSSUQ and SUS. The first one is a multidimensional questionnaire containing 19 items measuring the overall satisfaction, the system quality, the information quality, and the interface quality. On the other hand, SUS is a unidimensional, 10 items questionnaire.

Tullis and Stetson [57] studied five different questionnaires, SUS, QUIS, CSUQ (a variant of PSSUQ) [41], the Microsoft's Product Reaction Cards [3], and a questionnaire created by themselves. The authors asked 123 people to compare two different websites. They had to complete the same questionnaire, which was randomly assigned, for both websites. In their results, they found out that each questionnaire gave the better result for the first website. To determine which questionnaire would have the best accuracy, they analysed sub-samples of data at group size 6, 8, 12 and 14 and performed a *t-test* to determine if the first website was better than the second, that is, are the results of the sub-samples of data in accordance with the results of the overall study? It turns out that SUS's accuracy increases quicker than the other questionnaires. This means that if we increase the group's size, SUS is the fastest to converge on the final correct conclusion.

The reliability of SUS has been studied with respect to the coefficient alpha. In the last study, Lewis and Sauro [42] calculated a reliability of 0.92. Moreover, according to Landauer [38], a reliability which stands between 0.7 and 0.8 is acceptable for research. For all these reasons, and because it contains only 10 assessments compared to the 19 of PSSUQ, we think that SUS is an ideal candidate for our satisfaction questionnaire.

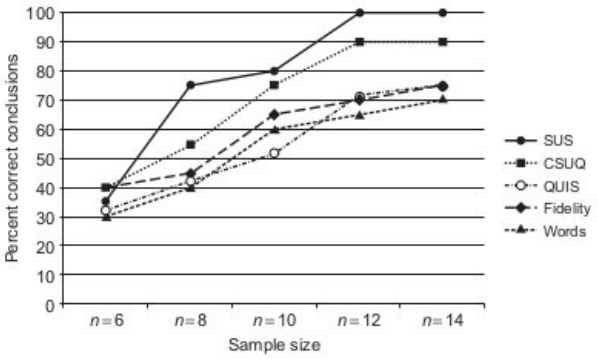


Figure 5.3 : Reliability comparison between five satisfaction questionnaires. Source – Sauro [30].

Experimental Procedure

The experimental procedure has to follow the same structure for each experiment. Doing so does not advantage or disadvantage certain subjects. The same explanations are given to all the subjects. At the beginning of the experiment, a short presentation of the procedure is given:

Goal We explain what the subject has to do: the scenario of the experiment.

Technical We review the six commands that are at his disposal.

Selection We show how the subject can know that a group is selected and how the group selection changed.

Split and Merge Two short videos are shown to describe how the split and the merge work. We insisted on the fact that the subject had to stop the split and that no obstacle could be between the groups while doing a merge.

Task Zone We explain what is a task zone and how the subject could know when the robots started working in the task zone.

Kinect Move Using a video, the user could understand how to steer the robots in the arena.

Recall of the Goal We finish the presentation by recalling to the user what he is going to achieve

The subjects then started the experiment either with the gesture-based control system or with the common graphical interface that uses the mouse. We have to pay attention to the order with which the system are tested by the users. Indeed, due to the carryover effect [30] if the same product is firstly tested, it may run the risk of unfairly biasing users either for or against this system and thus induce bias in the results. To address this potential problem, we alternate the first and the second system during the experiments: half the subjects started with the gesture control system and the other half started with the graphical user interface. Doing so will clearly not remove the potential preference of one of the systems. However, it will tend to cancel this effect when we aggregate the data across all the subjects.

5.3 System Feasibility Confirmation

The methodology proposed in section 5.2.1 aims to determine if the gesture-based control system is effectively usable for controlling several swarms of robots. This section shows preliminary high-level results from a small number of experiments.

We asked 18 people to perform the experiments described above. Figure 5.4 depicts the histogram of the user satisfaction questionnaire and the histogram for the total duration time for both of the interaction systems.

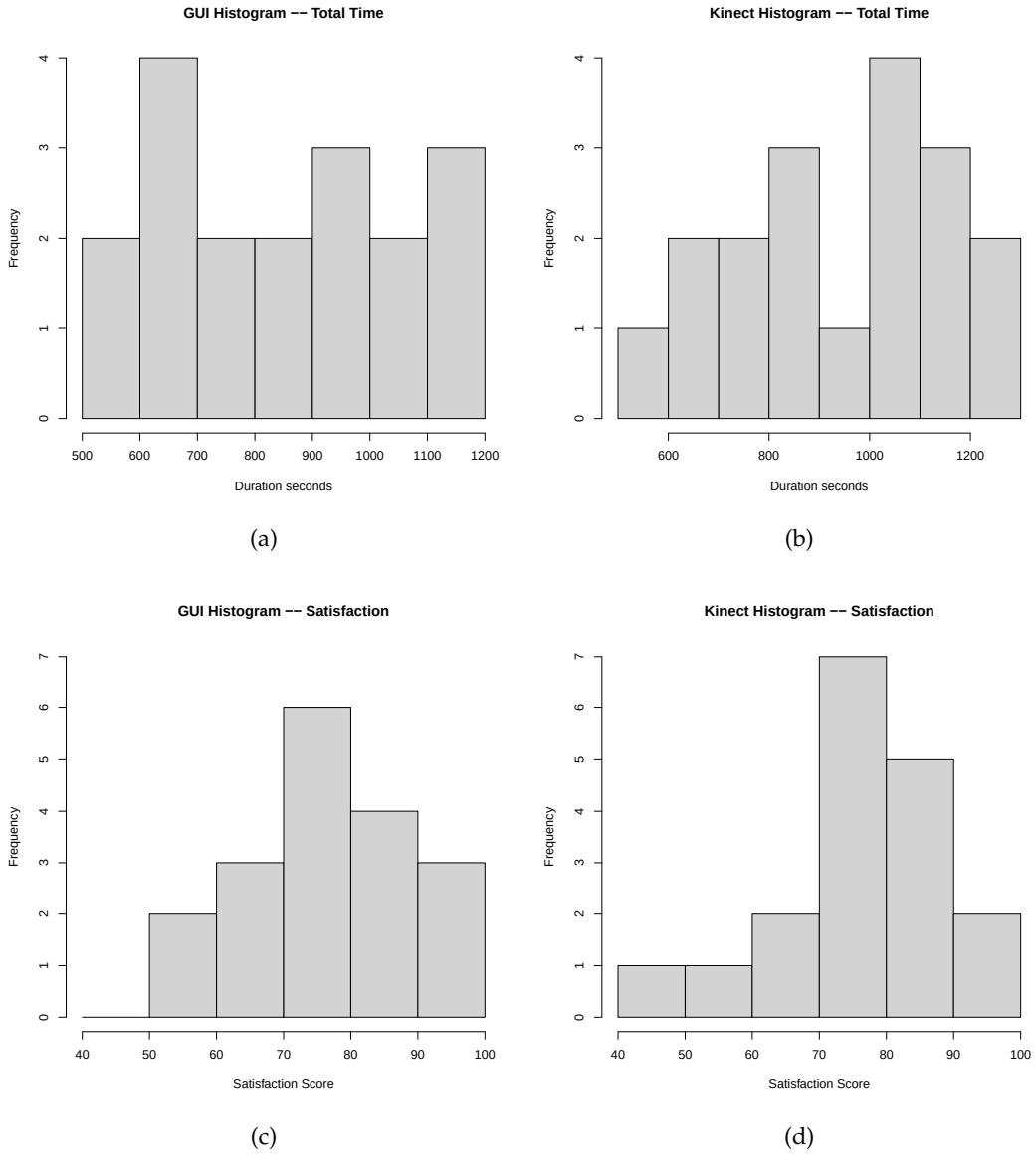


Figure 5.4 : Histograms: (a) frequency of the duration time for the graphical interface. (b) frequency of the duration time for the Kinect. (c) frequency of the user satisfaction for the graphical interface. (d) frequency of the user satisfaction for the Kinect.

First and foremost, we can affirm [1] that people did succeed in the experiments with the Kinect-based interaction system developed as part of this work. From a unique group of robots, they were capable to divide it in three sub-groups and to move them in three different zones of the arena using the steering direction mechanism.

Histogram 5.4(d) shows that a large amount of these 18 people are quite satisfied to use the Kinect for manipulating the swarms. Histograms 5.4(a) and 5.4(b) show that even if they could achieve their goal, they seem to be quicker using the graphical user interface [2]. However, a disadvantage of using the GUI is that it requires a lot of materials compared to the Kinect. Indeed, when using a graphical interface, the user must be facing a screen and using the mouse and keyboard. On the other hand, the Kinect system avoids these requirements and lets the user doing his gestures for controlling the robots.

Gesture recognition with the Kinect is still a young research field. For sure, improvements must be performed to reach, at most, the same performances as those with usual input devices. People are used to use mouse and keyboard to perform incredible actions. However, as these preliminary results show, Kinect can replace the common pair of mouse and keyboard with a clear advantage regarding the materials requirement.

5.4 Discussion and Conclusions

The aim of this chapter was to provide a detailed framework to perform usability tests for human to swarm interaction devices. We have designed a simple scenario which allows to extract different kind of data, such as time-on-tasks and satisfaction information.

We have tested our methodology on 18 people and it appears that using a gesture-based control system allows the subjects to manipulate swarms of robots. Even if these 18 people were not as quick as using the graphical interface, the Kinect device has been used without major difficulties.

We argue that using a Kinect device avoids the requirement of using a relatively big infrastructure to communicate with the robots. Indeed, a simple camera like the one developed by Microsoft is enough. However, this camera should still be connected to a computer. One can imagine in the near future that these kind of devices will be able to send their data through a wireless network. This will provide

1. See Chapter 6 for a statistical analysis of these results
2. We insist on the fact that these results does not mean that in average, people are generally quicker with the graphical interface. We suggest the reader to read Chapter 6 for more details.

greater flexibility in the use of the system.

Some usability criticisms have been gathered from the 18 experimenters though. No matter the system used, most of the subjects complained about the visual feedback of the robots. The bigger difficulty they faced was to figure out what was the robots' direction. People usually did not know where the robots where looking. Consequently, they had some pain to move the robots in the arena.

Based on this remark, an improved version of the simulation has been created. An arrow is drawn on the top of each robot. The arrow points in the direction of the robot's direction. To avoid differences in the difficulty of the experiments, the improved version of the simulation has not been tested by the subjects. Figure 5.5 shows the versions with and without the direction arrow.

(a) (b)

Figure 5.5 : (a) the robots have no direction arrow. (b) the direction arrow is drawn on the top of each robot.

Chapter 6

Statistical Comparison Tools

In this chapter, we present statistical tools for analysing the data collected during the experiments. We show several statistical tests that can be useful to study the difference of use between two interaction devices.

6.1 Statistical Tests

The goal of the experiment described in the previous chapter was to answer the question: *Is the Kinect better to use than the graphical user interface?* We decided to work with two different axis to measure this usability: the satisfaction of the users, and the time they take to complete the experiment. The System Usability Scale questionnaire gave a number between 0 and 100 and the time was measured in second. From these data, we now have to analyse the results in order to answer the question.

In usability tests, like in a large number of experiments, we do not have access to the entire population. A sample population is thus used to achieve estimations. In this case, the results obtained for a certain sample may not be the same for the entire population. It can even be different for another sample coming from the same population. Because it is clearly not possible to test all the population, we need statistical tests to answer the question with a known degree of error.

Generally, a statistic test uses a reductio ad adsurdum. In order to determine if a difference exists between two systems, we will suppose that there is no difference at all between the systems. This hypothesis is called the *null hypothesis (H0)*. The alternative hypothesis is called $H1$. The test then calculates the probability to observe the results under the null hypothesis. This probability is called the *p-value*. If this probability is less than α, a determined significance level, the null hypothesis is rejected. In other words, α is the risk to reject $H0$ while it is true. This kind of

mistake is known as the Type I Error. The usual values of α are 0.05 or 0.01. There also exists a Type II Error, which is the error made by accepting H0 if it is false and is denoted by β.

There exist several statistical tests, and the choice of one of them depends on the nature of the data and the power of the test. The power of a test is given by the probability of not committing a Type II Error.

Two different kinds of statistical tests exist: two-tailed test and one-tailed test. The first is used when the alternative hypothesis does not require us to specify the direction of the difference. The second one is used when the direction of the alternative hypothesis matters. More formally:

$$\text{Two-tailed test} \begin{cases} H0 & : & \theta_1 = \theta_2 \\ H1 & : & \theta_1 \neq \theta_2 \end{cases}$$

$$\text{One-tailed test} \begin{cases} H0 & : & \theta_1 = \theta_2 \\ H1 & : & \theta_1 < \theta_2 \text{ (one-tailed down)} \\ H1 & : & \theta_1 > \theta_2 \text{ (one-tailed up)} \end{cases}$$

where θ defines the observation parameter of the test. As part of this work, we are not interested in knowing if there exists a difference between the use of the Kinect and the use the GUI. What we are interested in is to know if the Kinect is better to use than the GUI. It appears from these definitions that one-tailed test is the most appropriate test.

6.1.1 Parametric or Non-parametric Test?

A parametric test requires strong constraints, so that the sample data has a normal distribution and the variance of different groups are the same. These constraints are not always satisfied and the smaller the data set is, the more difficult it is to verify these assumptions. On the other hand, a non-parametric test does not impose any distribution constraints. It is valid for relatively small samples and can compare distributions with different variances.

From the previous explanations, it is tempting to use only non-parametric tests. However, if the parametric tests' conditions are satisfied, they are more powerful than any non-parametric tests. That is, the probability P(reject $H0$ | $H0$ is false) is higher.

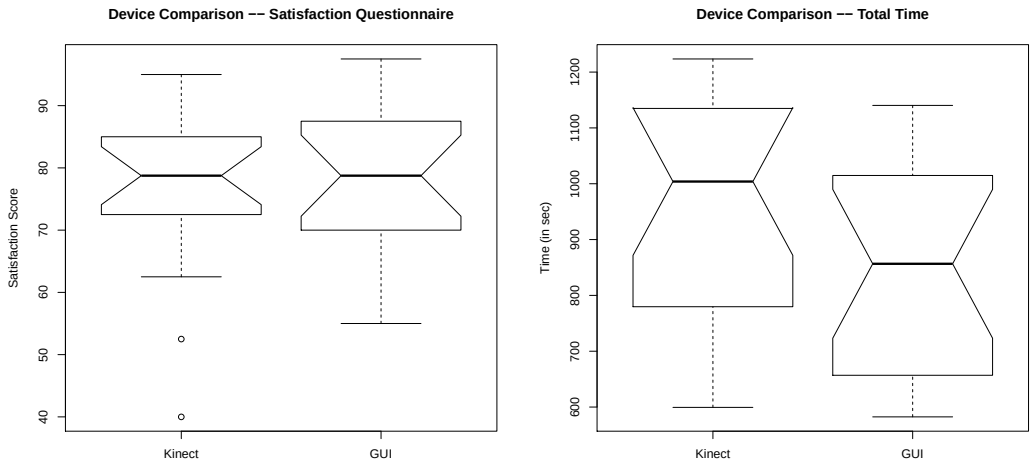
In usability studies, especially for those which focus on different kind of human-interaction devices, the experiments take a long time to proceed. Consequently, it is not always possible to gather a high amount of data. For example, the experiment described in the previous chapter takes on average one hour per subject without

taking into account the time spent to analyse the data. For this reason, and because we were not able to do more tests for the systems developed in this work, we decided for the remainder of this document, to focus only on non-parametric tests.

6.1.2 Comparing the Devices

Remember that we are interested in comparing the ease of manipulation of the Kinect device over the graphical interface. The following tests will then focus on this difference.

As part of this work, we have conducted 18 experiments. From these 18 subjects, 9 have started with the Kinect and the other 9 with the graphical user interface. Figure 6.1 depicts the boxplots of the satisfaction questionnaire and the total time spent by the subjects to achieve the entire experiments with each device. We can observe in these two boxplots that the notches of the boxplots overlap. Consequently, no statistically significant results can be concluded from the boxplots.



(a) User satisfaction boxplot.

(b) Boxplots of the total time.

Figure 6.1 : Notched Boxplots: (a) User satisfaction comparison. The medians of the two devices are the same but there is an overlap with the two notches. (b) Total time taken by the user to achieve the experiment with the two devices. We can see that the median of the Kinect is higher than the one of the GUI. However, the two notches also overlap.

Wilcoxon Signed-Rank Test

This test is the non-parametric alternative of the known *t-test* for correlated samples. Correlated samples, or *paired* samples are groups in which the subjects can be matched as pairs. This is indeed the case studied in this section. A set of 18 people have experimented with both the Kinect device and the graphical interface. The data resulting from the Kinect's experiments will be compared to the data resulting from the graphical user interface.

The program used to run the test is `R` which provides an implementation of the paired, one-tailed version of the Wilcoxon test. Table 6.1 presents the data on which the Wilcoxon test has been performed. Unfortunately, as we expected from the analysis of the boxplots, and due to the very small sample size, the p-value returned by the test was very high. Consequently, the null hypothesis could not be rejected in favour of the alternative.

Subjects	Total Time (sec)		Satisfaction Questionnaire	
	Kinect	GUI	Kinect	GUI
1	1215.5	1104.4	52.5	55
2	1134.9	1062.3	77.5	90
3	1172.4	966.9	62.5	75
4	998.1	1124.2	95	97.5
5	629.1	656.9	80	75
6	599.4	615.2	72.5	80
7	1135.3	582.4	40	75
8	844.2	982.2	65	77.5
9	758.3	694.2	82.5	87.5
10	655.2	832.4	75	92.5
11	1088.2	718.6	85	65
12	1025.6	587.4	85	70
13	1223.5	1140.3	87.5	87.5
14	1043.2	727.2	92.5	95
15	1009.8	727.2	75	55
16	897.6	626	80	85
17	779.7	979	82.5	70
18	834.2	880.8	75	80

Table 6.1 : For each subject: the total time in seconds for both systems and the SUS questionnaire score.

Mann-Whitney Test

We saw that due to the small sample, no significant statistical conclusions can be extracted from the Wilcoxon test. With a bigger sample however, it is likely that the test would be more significant.

The sample of 18 people was created by regrouping three sub-categories: (i) roboticists, (ii) computer scientists, and (iii) others (people that are neither roboticists nor computer scientists). We plan in the future to conduct more experiments with more people of each category. This will give the possibility to compare the usability of the three groups: are the roboticists quicker than computer scientists with the Kinect? Are the other users more satisfied to use the graphical interface than the roboticists? Time-on-tasks data have also been collected during the experiments. It is therefore possible to compare groups with respect to these values.

Comparing groups with each other can not be achieved by using the Wilcoxon Signed-Rank test because the sample are not paired anymore. The non-parametric version of the *t-test* for independent samples is the Mann-Whitney test, also called Wilcoxon Rank-Sum Test.

Applying the test to the very small data sets will not give any significant results. In the future, with more data the Mann-Withney test can be used.

6.2 Discussion and Conclusions

The aim of this chapter was to provide statistical tools and background to perform significant data analysis. The data collected during the experiments must be analysed through powerful statistical tests in order to have an objective opinion about human-swarm interaction devices.

We have tested our methodology on 18 people divided into three sub-categories. Due to this small sample, we have not been able to extract significant data regarding the systems developed in this work. However, we have provided all the tools needed to conduct strong analysis with more data.

Chapter 7

Conclusion and Future Work

The main goal of this thesis has been to develop a new system that allows a human to control and interact with swarms of robots. Few studies to date have focused on human-swarm interaction despite the importance of this field for future real applications of swarm robotics. This thesis provides the first steps in order to achieve interaction between human beings and swarms.

The first phase was the high-level analysis of the system. It consisted of determining the users requirement in terms of interaction with the swarms. For swarm robotics to be useful to a human, he has to be able to guide a swarm in a complex environment. When the robots are in the right place, the user can send his orders to the robots. To be able to guide swarms in a complex environment, we determined the necessary commands the user will have at his disposal, such as moving the swarms in the environment, split a swarm into two sub-swarms or merge two groups to form a bigger one. Finally, we determined the level of control of the operator for each command, that is, the frontier between robots self-decision and the precision of control of the operator. The second phase of the work was the implementation of the analysis phase and the development of the gesture recognition system.

We verified the feasibility of the system with several volunteers that performed simulations. A scenario in which the users had to control and guide multiple swarms of robots was created. The users who tested the system with this scenario were capable of controlling and guiding the swarms in order to achieve the tasks they were assigned.

A second contribution of this thesis is the development of a statistical framework allowing researchers to compare different kind of interaction devices. We chose gestures for communicating with the robots because we believe it has the advantages of being a natural communication tool (i.e. the steering mechanism using a virtual wheel for guiding the swarm). However, many different interaction systems may

be used, such as the very common couple mouse and keyboard, but also a joystick or even a smartphone. The statistical framework can be used to compare all these systems in order to study their advantages and disadvantages. This framework takes into account objective usability measures and subjective user preference.

Future Work

The system developed in this thesis provides a fully functional solution for interacting and controlling swarms in a complex environment. Nevertheless, human-swarm interaction is a relatively young field of research and many improvements can be made at different levels.

An important aspect of such an interaction system is the feedback the robots provide regarding their current state. In this thesis, we conducted experiments using a simulator. The simulator allowed us to give feedback to the user that would be unrealistic in a real world setting. When the robots received a new command, the name of this command was written on the simulator screen. This feedback assured the user that his command was correctly interpreted by the robots. In a real scenario, how could the operator be sure that the robots correctly received the command? The robots could use a colour-based feedback system. This could lead to some confusion. Indeed, we already use colours in two cases. First, each group has a different colour, which allows the user to visually separate the groups. We also use colour rules for selected groups. When the user selects a group in order to control it, it becomes yellow. If the users wants to merge two groups, the second selected one becomes red. A second use of the simulator capabilities is the possibility to dynamically draw geometric shapes in the arena. A remark which was often made was the confusion about the direction of the swarm while the user guided it. To address this issue, we modified the system in such a way that the simulator draws an arrow on the top of each robot. This arrow points in the robot's direction. In real circumstances however, another method should be used. We could add a direction information on the top of each robot. For instance, we could attach perpendicularly to the turret an extra arrow pointing in the direction of the robot. Unfortunately, this solution is too restrictive because each new robot should be modified. Moreover, the solution should be independent of the robot's structure. Further research can be done in the formation of a shape, giving the direction information and created by each robot belonging to the moving group. For example, each robot could position itself relatively to the other in order to form a big arrow, pointing to the group direction (see Figure 7.1).

A second field of improvement concerns the interaction device. We demonstrated that using gesture recognition allows the user to control the robots. However, even if gestures have some advantages like the steering wheel mechanism, some commands may not be easily associated to a gesture. For example, the selection gesture that

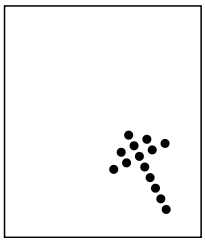
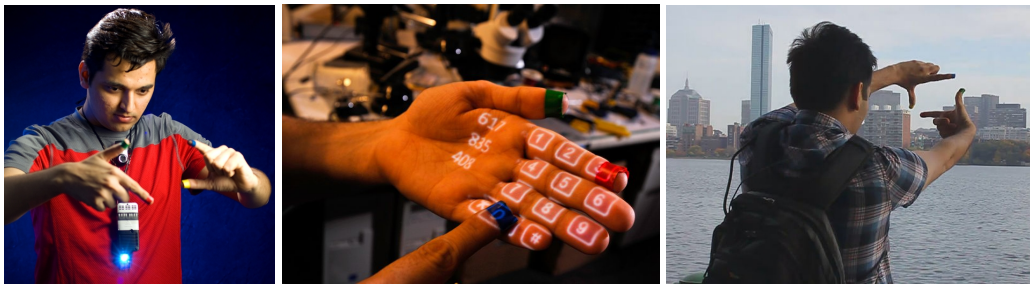


Figure 7.1 : Each robot is positioned in such a way that the swarm create an arrow shape that points to swarm direction.

we used in this work is not very intuitive. Because the system is designed so that the user cannot point at the group he wants to select, it is more difficult to find a relevant gesture. In the future, mixing different interaction system may turn out to be more efficient. Natural language can be one of the candidates in the perspective of not using any physical device (i.e. a mouse, a Wiimote, a smartphone, ...). Indeed, pronouncing the word *select* (even in many different languages) is clearly more intuitive than any kind of gestures. On the other hand, guiding a swarm in an environment is more difficult with natural language than with a steering wheel.



(a) (b) (c)

Figure 7.2 : The SixthSense device: (a) a camera and a projector is tied around the neck. (b) the device may project an image on any kind of surface and it is able to recognize the interaction with that image. (c) P. Mistry delimiting the zone with its hands to take a picture.

Regarding the gesture recognition as it is implemented in this work, the system still needs a camera connected to a computer and the human operator must face it. This system is not efficient if the user wants to control his robots in many different places. A new device was created by Pranav Mistry, working at the MIT Media Lab. The system removes the dependency to the screen, the mouse, the keyboard and even to any fixed physical workstation. This project, called SixthSense [1] is a small mobile device, comparable to a mobile phone and capable of recognizing user's

1. `http://www.pranavmistry.com/projects/sixthsense`

gestures (see fedigure 7.2). It can also be used for *augmented reality* adding virtual information on existing physical objects.

This system may be the solution for the main problems we noticed. Thanks to the augmented reality, information related to the robots state (swarm direction, commands acknowledgements, working status, . . .) may be virtually added in the environment. Moreover, the system removes the dependency of any Kinect-like camera connected to a computer.

Appendix A

System Usability Scale

Age :

Name :

Category : lambda hight-tech roboticist

Please, answer this questionnaire with the **control system** in mind. Try to answer your immediate response to each item. All item should be marked, if you feel that you cannot respond to a particular item, mark the center point of the scale.

I think that I would like to use this system frequently.	Strongly disagree				Strongly agree
	1	2	3	4	5
I found the system unnecessarily complex.	Strongly disagree				Strongly agree
	1	2	3	4	5
I thought the system was easy to use.	Strongly disagree				Strongly agree
	1	2	3	4	5
I think that I would need the support of a technical person to be able to use this system.	Strongly disagree				Strongly agree
	1	2	3	4	5
I found the various functions in this system were well integrated.	Strongly disagree				Strongly agree
	1	2	3	4	5
I thought there was too much inconsistency in this system.	Strongly disagree				Strongly agree
	1	2	3	4	5
I would imagine that most people would learn to use this system very quickly.	Strongly disagree				Strongly agree
	1	2	3	4	5
I found the system very cumbersome to use.	Strongly disagree				Strongly agree
	1	2	3	4	5
I felt very confident using the system.	Strongly disagree				Strongly agree
	1	2	3	4	5
I needed to learn a lot of things before I could get going with this system.	Strongly disagree				Strongly agree
	1	2	3	4	5

To calculate the SUS score, first sum the score contributions of the items 1, 3, 5, 7 and 9. The score contribution of these items are their scale position minus one. Then, sum the score contributions of the other items: five minus their scale position. Finally, multiply the sum of the scores by 2.5 to obtain the overall score that has a range of 0 to 100.

Bibliography

[1] Microsoft releases kinect for windows sdk beta for academics and enthusiasts. http://www.microsoft.com/en-us/news/press/2011/jun11/06-16MSKinectSDKPR.aspx, [Last accessed: April 19, 2012].

[2] Shishir Bashyal. Human swarm interaction for radiation source search and localization. *Swarm Intelligence Symposium, 2008. SIS 2008. IEEE*, 19(5):1 – 8, 2008.

[3] Joey Benedek and Trish Miner. Measuring desirability: New methods for evaluating desirability in a usability lab setting. *Proceedings of Usability Professionals Association*, 2003(03/04/2003):8–12, 2002.

[4] Gerardo Beni. From swarm intelligence to swarm robotics. *Swarm Robotics*, 3342:1–9, 2005.

[5] Nigel Bevan. Classifying and selecting ux and usability measures. In *Proc. of the 5th COST294-MAUSE Workshop on Meaningful Measures*, pages 13–18, 2008.

[6] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, Inc., New York, NY, USA, 1999.

[7] J Brooke. *SUS: A quick and dirty usability scale*, pages 189–194. Taylor and Francis, 1996.

[8] Rodney A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.

[9] DJ Bruemmer and DD Dudenhoeffer. Mixed-initiative remote characterization using a distributed team of small robots. *2001 AAAI Mobile Robot*, 2001.

[10] Robert Burton. Ivan sutherland. http://amturing.acm.org/award_winners/sutherland_3467412.cfm, [Last accessed: April 12, 2012].

[11] Gianni Di Caro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 1998.

[12] John P Chin, Virginia A Diehl, and Kent L Norman. *Development of an instrument measuring user satisfaction of the human-computer interface*, volume 218, pages 213–218. ACM, 1988.

[13] Lee Cronbach. Coefficient alpha and the internal structure of tests. *Psychometrika*, 16(3):297–334, 1951.

[14] Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. In Erol Şahin and William M. Spears, editors, *Swarm Robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 10–20. Springer, 2005.

[15] M. O. Anderson M.D. McKay. D. J. Bruemmer, D. D. Dudenhoeffer. A Robotic Swarm for Spill Finding and Perimeter Formation. *Spectrum*, 2002.

[16] J. W. Davis and M. Shah. Visual gesture recognition. *IEE Proc. Vision, Image and Signal Processing*, 141(2):101–106, 1994.

[17] Claire Detrain and Jean-Louis Deneubourg. Self-organized structures in a superorganism: Do ants "behave" like molecules? *Physics of Life Reviews*, 3(3):162–187, September 2006.

[18] Marco Dorigo and Erol Sahin. Guest editorial. Special issue: Swarm robotics. *Autonomous Robots*, 17(2–3):111–113, 2004.

[19] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. Bradford Company, Scituate, MA, USA, 2004.

[20] ENGELBART D.C. ENGLISH, W.K. and M.L BERMAN. Display-selection techniques for text manipulation. *IEEE Trans. Hum. Factors Electron*, pages 5–15, 1968.

[21] Xavier Ferré. Integration of usability techniques into the software development process. In *ICSE Workshop on SE-HCI*, pages 28–35, 2003.

[22] Ron George and Joshua Blake. Manipulations : Universal foundational metaphors of natural user interfaces. *Framework*, pages 1–5, 2010.

[23] A. Giusti, J. Nagi, L. Gambardella, S. Bonardi, and G. A. Di Caro. Human-swarm interaction through distributed cooperative gesture recognition. In *Proceedings of the 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI) (Video Session)*, Boston, MA, USA, March 5-8, 2012.

[24] Crina Grosan, Ajith Abraham, and Monica Chis. Swarm intelligence in data mining. *Intelligence*, 34(2006):1–20, 2006.

[25] Daniel Grünbaum, Steven Viscido, and Julia Parrish. Extracting Interactive Control Algorithms from Group Dynamics of Schooling Fish. In Vijay Kumar, Naomi Leonard, and A. Morse, editors, *Cooperative Control*, volume 309 of *Lecture Notes in Control and Information Sciences*, pages 447–450. Springer Berlin / Heidelberg, 2005.

[26] H. Rex Hartson. Human-computer interaction: interdisciplinary roots and trends. *J. Syst. Softw.*, 43(2):103–118, November 1998.

[27] Michael Isard and Andrew Blake. A mixed-state condensation tracker with automatic model-switching. pages 107–112, 1998.

[28] Nielsen J. Why you only need to test with 5 users, 2000. `http://www.useit.com/alertbox/20000319.html` [Last accessed : April 5, 2012].

[29] Nielsen J. Medical usability: How to kill patients through bad design, 2005. `http://www.useit.com/alertbox/20050411.html` [Last accessed : April 2, 2012].

[30] J. R. Lewis J. Sauro. *Quantifying the User Experience: Practical Statistics for User Research*. Morgan Kaufmann, 2012.

[31] Aleksandar Jevtić and Diego Andina. Swarm intelligence and its applications in swarm robotics. In *Proceedings of the 6th WSEAS international conference on Computational intelligence, man-machine systems and cybernetics*, CIMMACS'07, pages 41–46, Stevens Point, Wisconsin, USA, 2007. World Scientific and Engineering Academy and Society (WSEAS).

[32] Jeff Johnson, Teresa L. Roberts, William Verplank, David C. Smith, Charles H. Irby, Marian Beard, and Kevin Mackey. The Xerox Star: A Retrospective. *Computer*, 22(9), 1989.

[33] J. E. Jones. On the determination of molecular fields. ii. from the equation of state of a gas. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 106(738):pp. 463–477, 1924.

[34] Zsolt Kira and Mitchell A Potter. Exerting human control over decentralized robot swarms. *Autonomous Robots*, pages 566–571, 2009.

[35] Jurek Kirakowski and Mary Corbett. SUMI: the Software Usability Measurement Inventory. *British Journal of Educational Technology*, 24(3):210–212, 1993.

[36] Andreas Kolling, Steven Nunnally, and Michael Lewis. Towards human control of robot swarms. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, HRI '12, pages 89–96, New York, NY, USA, 2012. ACM.

[37] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Addison-Wesley Publishing Company, USA, 5th edition, 2009.

[38] Thomas K. Landauer. *Behavioral Research Methods in Human-Computer Interaction*, volume 28, chapter 9, pages 203–227. Elsevier, Amsterdam, 1997.

[39] Hyeon-Kyu Lee and Jin H. Kim. An hmm-based threshold model approach for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21:961–973, 1999.

[40] James Lewis. Ibm computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction*, 7(1):57–78, 1995.

[41] James R. Lewis. IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction*, 7(1):57–78, 1995.

[42] James R. Lewis and Jeff Sauro. The factor structure of the system usability scale. In *Proceedings of the 1st International Conference on Human Centered Design: Held as Part of HCI International 2009*, HCD 09, pages 94–103, Berlin, Heidelberg, 2009. Springer-Verlag.

[43] James R J R Lewis. *Psychometric evaluation of the post-study system usability questionnaire: The PSSUQ*, volume 36, page 1259–1263. Human Factors and Ergonomics Society, 1992.

[44] L.M. Gambardella F. Mondada S. Nolfi T. Baaboura M. Birattari M. Bonani M. Brambilla A. Brutschy D. Burnier A. Campo A.L. Christensen A. Decugnière G. Di Caro F. Ducatelle E. Ferrante A. Förster J. Guzzi V. Longchamp S. Magnenat J. Martinez Gonzalez N. Mathews M.A. Montes de Oca R. O'Grady C. Pinciroli G. Pini P. Rétornaz J. Roberts V. Sperati T. Stirling A. Stranieri T. Stuetzle V. Trianni E. Tuci A.E. Turgut F. Vaussard M. Dorigo, D. Floreano. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics and Automation Magazine*, page in press, 2012.

[45] Ritch Macefield, Cornovian Close, and Wolverhampton Wv Nu. How to specify the participant group size for usability studies: A practitioner's guide. *Journal of Usability Studies*, pages 34–45, 2009.

[46] Mick McGee. Master usability scaling: magnitude estimation and master scaling applied to usability measurement. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '04, pages 335–342, New York, NY, USA, 2004. ACM.

[47] James McLurkin, Jennifer Smith, James Frankel, David Sotkowitz, David Blau, and Brian Schmidt. Speaking Swarmish: {Human-Robot} Interface Design for Large Swarms of Autonomous Mobile Robots. 2006.

[48] Sushmita Mitra and Tinku Acharya. Gesture recognition: A survey. *IEEE Transactions on Systems, Man and Cybernetics - Part C*, 37(3):311–324, 2007.

[49] J C Nunnally. *Psychometric Theory*, volume 3. McGraw-Hill, 1978.

[50] Helen Petrie and Nigel Bevan. The evaluation of accessibility , usability and user experience. *Office*, pages 299–315, 2009.

[51] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Timothy Stirling, Álvaro Gutiérrez, Luca Maria Gambardella, and Marco Dorigo. ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, pages 5027–5034. IEEE Computer Society Press, Los Alamitos, CA, September 2011.

[52] PrimeSense Inc. *Prime Sensor^{TM} NITE 1.3 Algorithms notes*, 2010. Last viewed 10-25-2011.

[53] Alain Rame and Sylvie Therond. *Anatomie et physiologie*. Elsevier Masson, 2006.

[54] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics*, pages 25–34, 1987.

[55] Ivan E. Sutherland. Sketch pad a man-machine graphical communication system. In *Proceedings of the SHARE design automation workshop*, DAC '64, pages 6.329–6.346, New York, NY, USA, 1964. ACM.

[56] W. Albert Th. Tullis. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. 2008.

[57] Stetson Tullis. A comparison of questionnaires for assessing website usability. In *Proceedings of the Usability Professionals Association (UPA) 2004 Conference*, pages 7–11, June 2004.

[58] Ali E. Turgut, Hande Çelikkanat, Fatih Gökçe, and Erol Şahin. Self-organized flocking with a mobile robot swarm. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 1*, AAMAS '08, pages 39–46, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[59] Andries van Dam. Post-wimp user interfaces, 1997.

[60] Alan Woolrych and Informatics Centre. Why and when five test users aren't enough. In *Proceedings of IHM-HCI 2001 Conference*, pages 105–108, 2001.

[61] F. Zijlstra. *Efficiency in work behavior. A design approach for modern tools.* PhD thesis, Delft University of Technology. Delft, The Netherlands: Delft University Press., 1993.

List of Figures

List of Algorithms