

MÄLARDALENS HÖGSKOLA
Department of Computer Science and
Electronics



FINAL THESIS:

Simulation of Bluetooth™ Scatternet for battery
life optimization

Author: David Espinosa Alfaro.
Supervisor: Javier García Castaño.

TABLE OF CONTENTS

TABLE OF CONTENTS

1. TABLE OF CONTENTS.....	4
2. REPORT.....	6
2.1. INTRODUCTION.....	7
2.1.1. Presentation.....	7
2.1.2. Bluetooth™ technology.....	8
2.1.2.1. Introduction.....	8
2.1.2.2. The Bluetooth™ name and history.....	8
2.1.2.3. The Bluetooth™ SIG.....	9
2.1.2.4. Master, Slave and Piconet.....	10
2.1.2.5. The Bluetooth™ nets: piconets and scatternets.....	10
2.1.2.6. RSSI.....	11
2.1.2.7. Bluetooth™ transmission protocol	12
2.1.3. Summary of the project.....	15
2.2. THE PROBLEM.....	16
2.2.1. The battery problem in portable devices.....	16
2.2.2. Battery consumption in Bluetooth™ technology.....	16
2.2.3. No algorithm defined in the Bluetooth™ specification.....	17
2.3. THE SOLUTION.....	18
2.3.1. Introduction.....	18
2.3.2. The algorithm.....	24
2.3.2.1. Considerations.....	24
2.3.2.2. Static and Dynamic Algorithm.....	28
2.3.2.3. Current Algorithm.....	30
2.3.2.4. Static Algorithm for one piconet.....	33
2.3.2.5. Dynamic Algorithm for one piconet.....	36
2.3.2.6. Static Algorithm for a scatternet.....	39
2.3.2.7. Dynamic Algorithm for a scatternet.....	46
2.3.3. Software.....	50
2.3.3.1. Used software tools: Lab Windows.....	50
2.3.3.2. The program.....	52
2.3.3.2.1 Main Part.....	53
2.3.3.2.2 Algorithm Part.....	57
2.3.3.2.3 Random Part.....	62
2.4. RESULTS.....	65
2.4.1. Prolongation of the life of the piconet/scatternet.....	66
2.4.2. Optimization of the consumption of the Bluetooth™ devices.....	69
2.4.3. Optimization of the piconet/scatternet configurations.....	75
2.5. FUTURE STEPS.....	77
2.5.1. Other factors to consider.....	77
2.5.2. Hardware experimenting.....	79
2.6. CONCLUSSIONS.....	82
2.6.1. About the results.....	82

2.6.2. Need to verify the results of the simulation.....	82
2.6.3. Keep on working in the future.....	82
3. PLANES.....	83
3.1. main.c code.....	85
3.2. definitions_interface.h code.....	107
3.3. main_algorithm.h code.....	110
3.4. definitions_algorithm.h code.....	122
3.5. 2_piconet.h code.....	125
3.6. 3_simulate.h code	140
3.7. main_random.h code	146
3.8. definitions_random.h code.....	148
3.9. 1_piconet_random.h code.....	149
4. USER'S MANUAL.....	153
4.1. Introduction.....	154
4.2 Graphical User Interface.....	155
4.3. Example of use.....	159
5. BIBLIOGRAPHY.....	161

REPORT

2.1. INTRODUCTION

2.1.1. Presentation

This project, *Simulation of BluetoothTM Scatternet for battery life optimization*, has been done in the *Mälardalens Högskola* (Sweden) within the Socrates-Erasmus exchange program. This is a research project about Bluetooth, a new wireless technology very used nowadays, and that is growing more and more everyday.

The Mälardalens University, which is located in the nice city of Västerås, is a perfect place to make a research project about Bluetooth, because they have been researching in this subject for several years and they have all the necessary resources to do it. In that way, a lot of articles and publications done by professionals of this university, can be found in the most popular and important Electronics magazines.

This project tries to solve one of the biggest problems in radio communications systems: the battery consumption of the devices. The report of this project is structured according to this way:

- First of all, it begins with an introduction to Bluetooth, where the most important points of this technology are analyzed. This introduction is really important for later understanding of the problem and the solution developed.
 - Next it comes a second part which constitutes the main part of the project: a deep analysis of the considered problem, the solution designed to solve this problem, and the results obtained with the simulations.
 - Finally, it is shown some future steps that may be done to complete this project, and the conclusions of the whole project.
-

2.1.2 Bluetooth™ technology

2.1.2.1 Introduction

Bluetooth™ has been the subject of much hype and media attention over the last three or four years. As various manufacturers prepare to launch products using Bluetooth™ technology, an unsuspected public is about to be catapulted into the next stage of the information technology revolution.

Bluetooth™ is a low cost, low power, short-range radio technology, originally developed as cable replacement to connect devices such as mobile phone handsets, headsets, and portable computers. This itself sounds relatively innocuous; however, by enabling standardized wireless communications between any electrical devices, Bluetooth™ has created the notion of a Personal Area Network (PAN), a kind of close range wireless network that looks set to revolutionize the way people interact with the information technology landscape around them.

No longer do people need to connect, plug into, install, enable, or configure anything to anything else. Through a ubiquitous standardized communications subsystem, device will communicate seamlessly. One does not need to know where one's cellular phone is, or even if it is switched on. As soon as the Web browser appears on the mobile computer screen, a link is established with the phone the Internet Service Provider is connected to, and the user is surfing the web.

The Bluetooth™ specification is an open and global specification which defines the complete system from the radio right up to the application level. The protocol stack is usually implemented in hardware and partly as software running on a microprocessor partly with different implementations partitioning the functionality between hardware and software in different ways.

2.1.2.2 The Bluetooth™ name and history

The name Bluetooth™ itself draws attention, mostly because of its unconventional nature. Most new industries are known by the name that describes their associated technology, however the Bluetooth™'s story is a bit different. There are different stories of the name Bluetooth™ and how that name came to be chosen. The general accepted story is presented below.

From approximately 940 A.D. to 985 A.D. Harald Blåtand was a Viking king of Denmark. During that time king, Harald achieved to unite Norway and Denmark. He also introduced Christianity to Scandinavia during his reign.

The translation of Blåtand is Bluetooth™. The name was adopted because Bluetooth™ wireless technology is expected to unify the telecommunications and computing industries, and also because Bluetooth™ technology was also originated in Scandinavia.

2.1.2.3 The Bluetooth™ SIG

The Bluetooth™ Special Interest Group (SIG) is a group of companies which work together to promote and define the Bluetooth™ specification. The Bluetooth™ SIG was founded in February 1998 by the core promoters Ericsson Mobile Communications AB, Intel Corporation, IBM Corporation, Toshiba Corporation and Nokia Mobile Phones.

In May 1998, the core promoters' public announced the global SIG and invited other companies to join the SIG as Bluetooth™ adopters in returning for a commitment to support the Bluetooth™ specification. The core promoters published version 1.0 of the Bluetooth™ specification in July 1999, the Bluetooth™ core enlarged with the addition of four more major companies, Microsoft, Lucent, 3COM and Motorola.

Any incorporated company willing to sign the Bluetooth™ SIG membership agreement can join the SIG as a Bluetooth™ adopter company. To join the SIG, companies simply fill in a form on the Bluetooth™ web site. This form commits SIG members to contribute with any key technology that can be needed to implement Bluetooth™.

This commitment linked to share technology means that Bluetooth™ SIG member companies who put their products through Bluetooth™ qualification are granted a free license to build products using the Bluetooth™ wireless technology. The license is important because there are patents required to implement Bluetooth™ ; companies that do not sign the Bluetooth™ adopter's agreement will not be entitled to use this technology. This offer proved so attractive that by April 2000, the SIG membership had grown to 1790 members.

Apart from giving a free license to patents, it is needed to implement Bluetooth™ wireless technology, Bluetooth™ SIG members also have permission to use the Bluetooth™ brand.

There are restrictions on the use of the brand, and these are set out in the Bluetooth™ brand book, the trademark may only be used on products where these ones can be correctly proved following the Bluetooth™ specification by completing the Bluetooth™ qualification program (a testing process).

To get the Bluetooth™ figure mark and instructions about the way of using it, companies sign up the Bluetooth™ trademark agreement.

2.1.2.4 Master, slave and piconet

Bluetooth™ devices can operate in two roles, as a master or as a slave. The master is usually the one that initiates the connections and establishes the frequency hopping sequence. Slaves synchronize to the master in time and frequency by following the master's hopping sequence.

A piconet is established when one or more slaves devices are operating together with one common master device. All the slaves in a piconet follow the frequency hopping sequence and timing of the master. In figure 1, it is possible to observe typical Bluetooth applications:

The master is able to communicate simultaneously with up to seven other slave devices. And many other devices could remain connected to the master in a parked state.

Bluetooth™ devices in a piconet support voice and data communications. For voice communications the Synchronous Connection Oriented link is used. For asynchronous data communications Asynchronous Connection Less links are required.

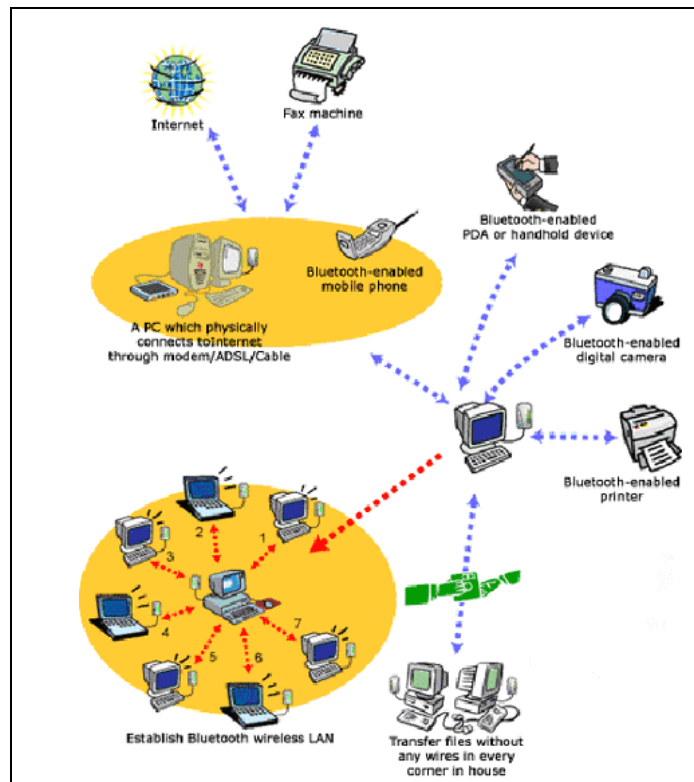


Figure 1: Bluetooth™

2.1.2.5 Bluetooth™ nets: piconets and scatternets

Bluetooth™ devices can work as master or as slave. The Bluetooth™ nets are called piconets, and they are made up by one master and several slaves (maximum seven devices working as slaves). The communication is always established between the master and one slave and never between two slaves. If one slave wants to communicate with another, it will first communicate with the master, and then the master will communicate with the other slave.

The piconets could be thought to be the only way to create Bluetooth™ nets. However, it is only possible to communicate with a maximum number of 8 devices. The solution for this problem is named **scatternet**, which is a group of piconets connected, using a bridge device (that belongs to two piconets).

The bridge device can be a device working as Master in one piconet and as Slave in the other one, or a device working as Slave in both piconets. Both situations can be seen in the figure below: piconets A and B are linked by a device that is working as master in piconet B and as a slave in A; piconet A and C are linked by a device working as a slave in both piconets . It is not possible to have a device working as Master in more than one piconet.

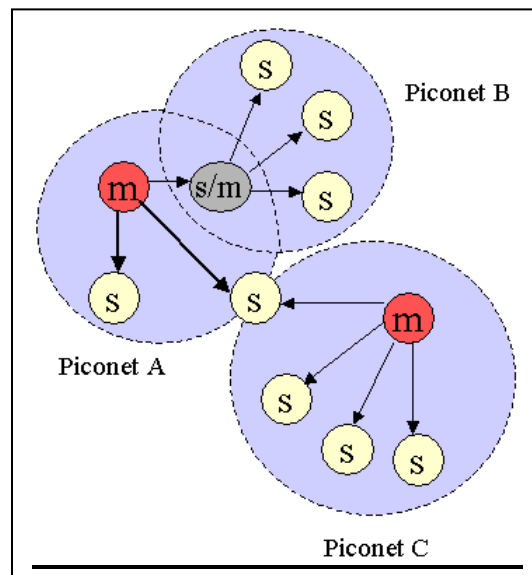


Figure 2: Bluetooth™ scatternet

2.1.2.6 RSSI

When the master is connected to a slave, it is necessary to know about the quality of the data communication. RSSI (Received Signal Strength Indicator) is an indication to the master from the received signal strength of the slave. This Bluetooth™ feature is used in the current project as the information needed to execute our algorithm.

To make sure that Bluetooth™ devices are communicating correctly, they have to operate inside of the golden range, which ranges from -60 dBm to -40 dBm.

2.1.2.7 Bluetooth™ Transmission Protocol

The **Physical Channels** uses two techniques at the same time:

- **Pseudo Random Frequency Hopping:** transmission rapidly hops among the available channels.
- **Time Division Duplexing (TDD):** Transactions are divided into dedicated time slots each for the Master and the Slave. Typically odd cycles for the Master and evens for the Slaves.

The transmission protocol that Bluetooth uses is Time Division Duplexing. This means that the available bandwidth is divided into 2 segments in the time domain, one for the network Master device transmissions, and another one for Slaves.

The major time domain construct in the Bluetooth protocol is the **FRAME**. Each frame contains 2 or more transmission **SLOTS** which are allocated in series to Master and Slave transmissions. Slots are each 625 microseconds in duration. The **FRAME** and **SLOT** terms are summarized below:

- **Frame** = a complete transmit/receive cycle
- **Slot** = a 625 microsecond segment within a frame

In figure 3, it is shown a transmission between a master and one slave, where the concepts explained are present:

- ✓ Complete packet transmission occurs during a Slot
- ✓ Frequency hops from Slot to Slot to Slot
- ✓ Frames define matched Master / Slave Slot transmissions

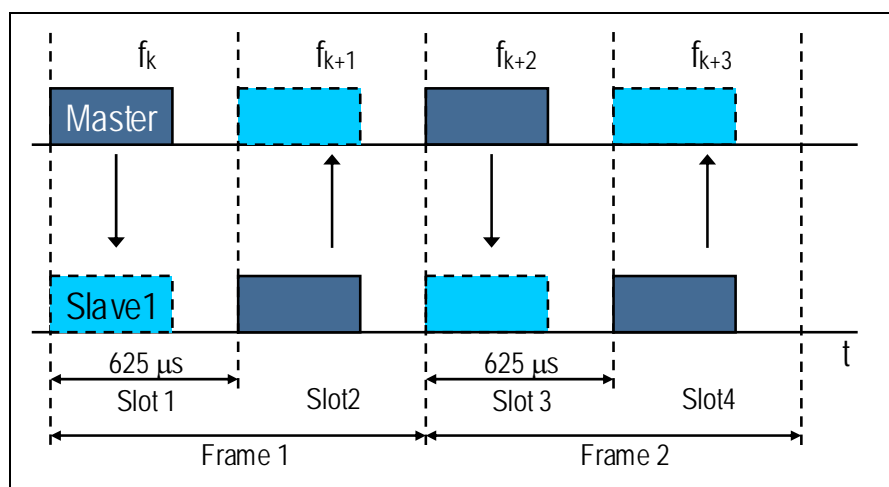


Figure 3: Frequency Hopping and TDD example

Apart from the previous example, where the transmission was just between a master and a slave, it can be found some other ways of transmissions that are shown following:

- **Multi-Slave Transmission:** The Bluetooth master interleaves traffic among multiple simultaneously active slaves. Each Master can support up to 7 simultaneously active slaves.

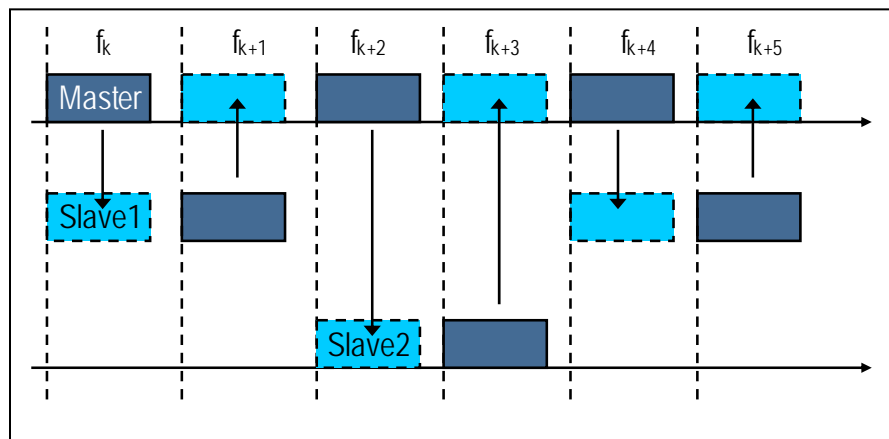


Figure 4: Multi-slave transmission

Bluetooth Masters can support up to 7 Active Members of a Piconet (the name for Bluetooth networks). Communication among multiple devices is interleaved as illustrated above, with the Master switching dialog from device to device as necessary, but always maintaining the same basic transmit/listen Framing model.

While up to 7 Active Slaves can be connected, only the Slave addressed in the Master's transmission participates in that particular Frame. Slaves with a different ID will simply ignore the transmission.

- **Multi-Slot Framing:** To increase performance Bluetooth supports a feature called Multi-Slot Frames in which several sequential slots are aggregated together to support a larger packet size. The example below is a 3/1 Frame in which the Master is allocated 3 sequential Slots and the Slave a single reply Slot. Bluetooth also supports 5/1 Framing which provides its maximum performance of 721Kbps in the fat channel and 57Kbps in backchannel. Multi-Slot Frames can be asymmetric in either direction.

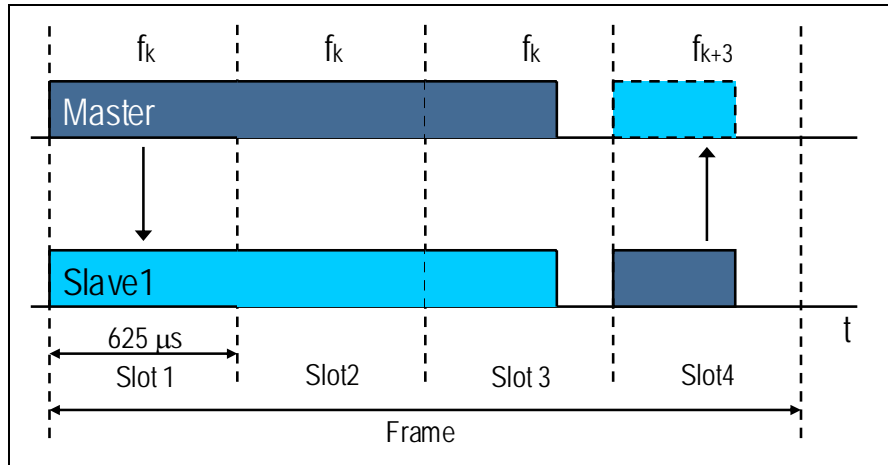


Figure 5: Multi-slot transmission

Multi-Slot Frames improve efficiency in several ways:

- ✓ They eliminate all header overhead from the 2-n Slot Payloads
- ✓ They double the raw bandwidth allocated to the fat channel by giving it consecutive Slots instead of interleaved Slots.
- ✓ Transmission across Slot boundaries continues where timing margins normally exist

- **Point to Multi-Point Transmission:** Bluetooth Master's can also support Point to Multi-point transmissions. In this case the Master addresses the transmission to ID 0. Slave's receiving a transmission in the Master's Slot with this ID know that this is a point to multi-point transmission and will process it. Note that no back channel traffic is allowed in point to multi-point transmissions.

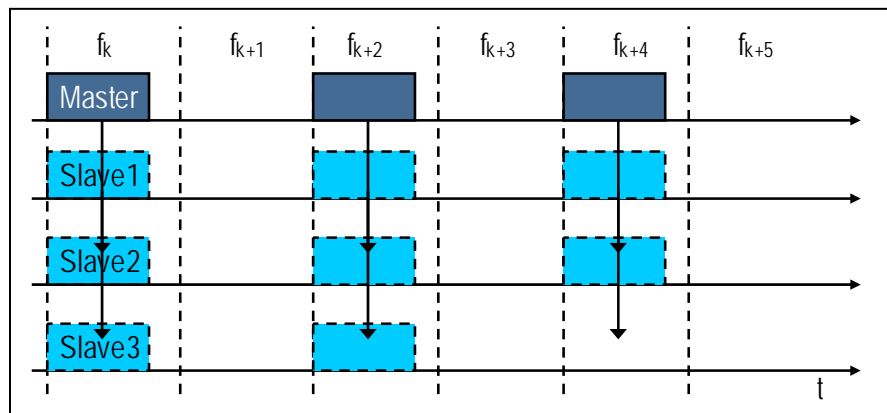


Figure 6: Point to Multi-point transmission

In this project, we consider a Multislave transmission, where all the devices have always something to transmit. Thus, this algorithm will be a generic one, but it will be possible to adapt it to another way of transmission (point to multipoint, multi-slot framing...).

2.1.3 Summary of the project

The Bluetooth technology is used to replace cables to connect small devices such as mobile phone handsets, headsets, mp3 players, computers, keyboards, mice... Most of these devices are portable, and then, they work with battery.

In this project, it has been developed a solution to optimize the battery life of the devices and to prolong their life too. This solution is an algorithm that serves for creating the configuration of the Bluetooth net (piconet and scatternet), in order to prolong the life of the whole configuration as much as possible.

2.2. THE PROBLEM

In the following, it is introduced the problem which has inspired this project: the battery consumption in Bluetooth™ devices and the possibility to increase the battery lifetime of the devices. Analyzing the problem, three important aspects must be present in mind:

2.2.1 The battery problem in portable devices

As said in the Introduction section, Bluetooth™ is a low power technology. According to it, it is possible to think that the consumption is not a critical aspect, and thus, there is not any problem to solve; however, Bluetooth™ is a wireless technology where most of the devices that work with it are portable, such as mobile phones, headsets, handsets, keyboards, mice, mp3 players... And, like everybody knows, one critical aspect of portable devices is the life of their batteries.

Having all these aspects in mind, it can be concluded that there is a need to make the best use of the batteries in the Bluetooth™ devices, in order to prolong their lives as much as possible.

2.2.2 Battery consumption in Bluetooth™ technology

Considering a multislave transmission, (the Bluetooth master interleaves traffic between multiple simultaneously active slaves), where all the devices, included the master, have always something to transmit, it is quite easy to see that the master of the piconet has a higher consumption than the slaves, because the master is always transmitting or receiving information, while each slave works when the time to do it comes.

This higher consumption of the masters provokes that these devices spend faster their batteries, and so their battery lives are quite shorter. Considering that the life of the configuration (piconet or scatternet) is the time since the configuration begins working till one device finishes its battery, it is easy to see that the life of the configuration will be the life of the first master who exhausts its battery, because the masters are the first devices that appear dying.

In this way, in normal Bluetooth™ behavior, considering multislave transmission, the configuration will usually die because of a master. And this is a big problem, because once the master is dead, the whole piconet is also dead because there is no way to communicate two slaves without master, and so all the slaves of the piconet have no way

to work. And the problem could be also more serious, if the dead piconet had been worked as a bridge between other two piconets: in that case there will be no way to communicate the devices of those two piconets.

Analyzing the battery left in the devices, when the configuration is dead, another problem is found: the devices that have been working as masters have spent or almost spent their battery, while the devices that have been working as slaves have almost all their battery left. This entire means that the configuration has dead when most of the devices (the slaves) had spent a really small part of their battery, and just a few ones had spent most of their battery (masters): a lot of battery is still left in the whole configuration.

2.2.3 No algorithm defined in the BluetoothTM specification

Finally, it must be said that nowadays there is no algorithm defined in the BluetoothTM specification to create the scatternets. This means, that scatternets are made in a random way, which causes illogical configurations that increase the battery consumption of the devices.

The no existence of a scatternet creation algorithm and all the battery problems explained before have motivated the search for a solution. This solution is an algorithm used to create a scatternet configuration, where a better use of the devices battery is obtained, and so it could be useful to prolong their life by a long time.

2.3. THE SOLUTION

2.3.1 Introduction

To solve the problem explained before it has been developed an algorithm to create scatternets, where the battery life optimization is the goal. That is the reason why a software model of simulation using Lab Windows (C Language) has been built. Unfortunately, it was not possible to test the algorithm with real Bluetooth devices, due to the lack of time.

To make this introduction easier to understand, a little example will be used in every point. This model of simulation works according to this way:

- The model just needs to know the number of devices and their position in space (coordinates).

Example: with 19 devices distributed in space like this

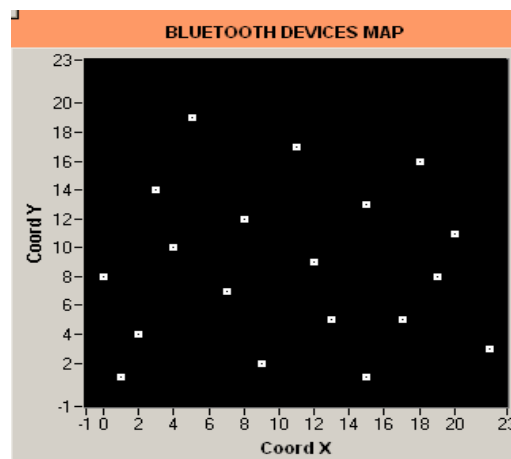
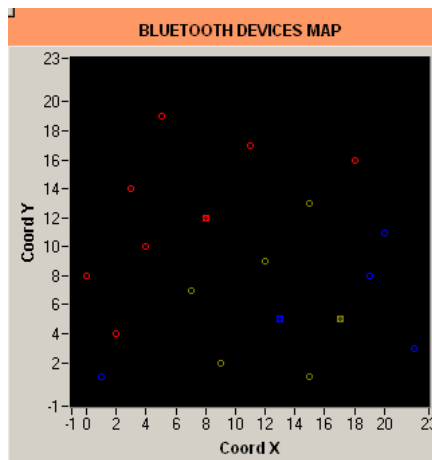


Figure 7: Initial map

- Just with this information, it will create and simulate three different scatternet configurations:
 - Create a **current scatternet configuration**: actually, there is no algorithm defined in the Bluetooth specification for scatternet formation. So to simulate the behavior of current scatternets, just a random one will be

created: a little algorithm has been written to create a valid and random configuration.

Example: it can be seen a random configuration that the model of simulation has built for this example. It is easy to see in the picture that the structure of the configuration is a bit illogical, the masters are in strange positions, most of the slaves are not connected to the closest master... Some factors that affect the battery consumption are found in a bad way.



CAPTION

- **Master of a Piconet**
- **Slave of a Piconet**
- **Bridge Slave**

Figure 8: Random Configuration

- Create a **static algorithm configuration**: once it has been developed and simulated the behavior of a current piconet/scatternet, it is the time of thinking about a different configuration where the life of the piconet/scatternet is prolonged as much as possible. To do that, it has been developed an algorithm which will indicate the best configuration: what the masters and what the slaves are, and the way of connecting the different piconets (if there are more than one).

Example: it is easy to see just in the picture, that the configuration is suitable: every slave is connected to the closest master, and the piconets are connected in a suitable way.

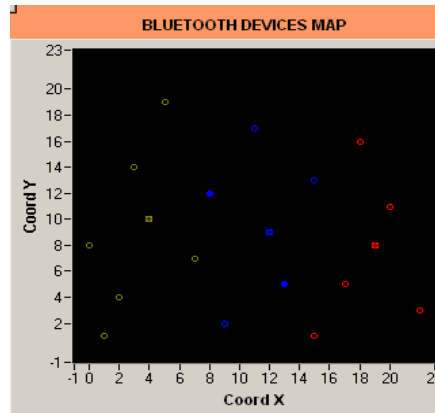


Figure 9: Static Algorithm Configuration

- Create a **dynamic algorithm configuration:** with the static configuration, the goal of prolonging the life of the scatternet was achieved. But, after some simulations, it was easy to realize that the life of the scatternet was conditioned by the masters: the masters consumption is higher than the slaves, and in this way their battery is empty sooner. That means that while the masters are empty or almost empty, most of the slaves have their battery almost full: the scatternet dies with a lot of battery remaining in most of the devices.

In order to fix this problem, it has been developed a dynamic algorithm trying to make better use of the battery linked to all the devices: instead of using a configuration until one of the devices is dead (empty battery), the configuration is changed when one device (normally a master) reaches a threshold battery value. With this configuration, it is prolonged the whole life of the piconet/scatternet, making a better use of the battery of all the devices.

Example: different configurations used to prolong the life of the scatternet

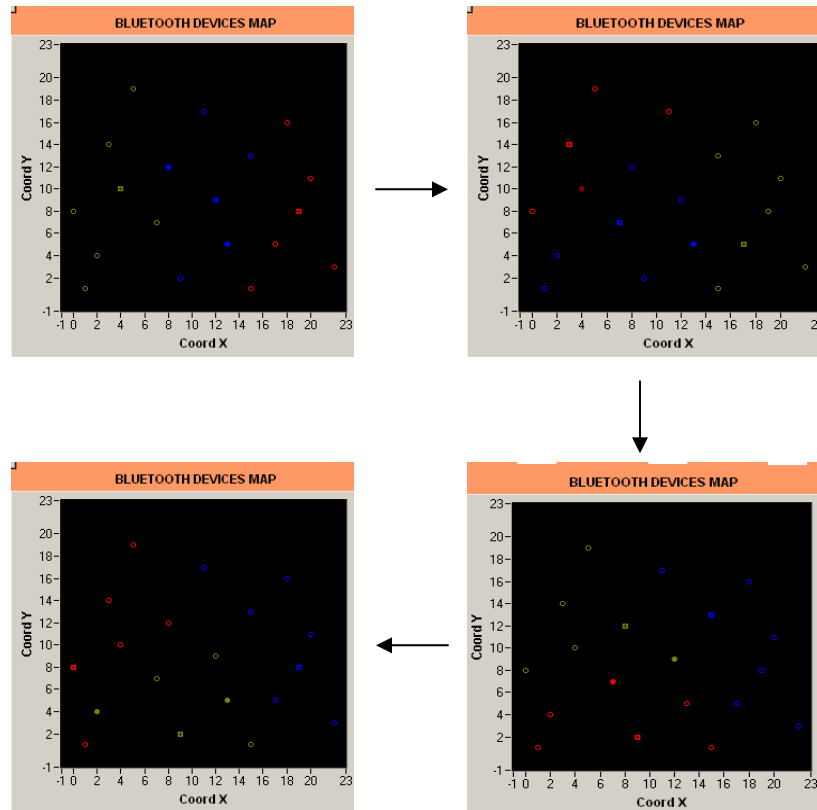


Figure 10: Four different configurations of the Dynamic Algorithm

- It will be simulated these three models: one simulates the current piconet/scatternet behavior, and the other two try to improve the results of the first one. For each model, the following results were obtained:

- ✓ **Life of the piconet/scatternet:** this means the time it takes since the piconet/scatternet begins working, till one device has no battery left.

It could be thought that when one device has no battery left, the configuration could go on working normally; however, as it has already been explained, the devices that spend faster their battery are the masters, and if one master is not working, all the devices of that piconet will not be able to work either, and maybe that piconet was a piconet bridge between two other ones,....

Example: comparing the life of each configuration. It is easy to see how the life of the static algorithm is around 70% higher than the current, and the dynamic is around 300% higher.

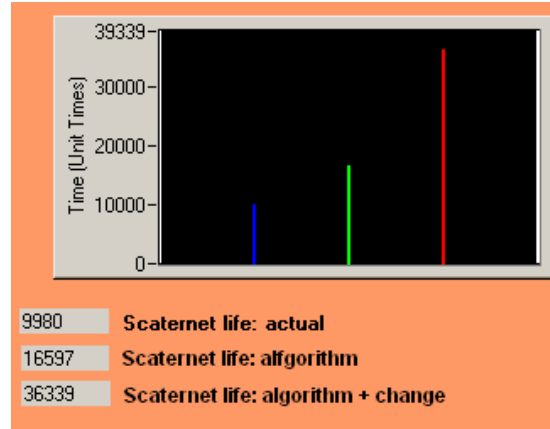


Figure 11: Scatternet Lives

- ✓ **Battery evolution along time of every device:** the consumption of each device along time is kept, so it can be seen the different values got for a device working as a slave, and for a device working as a master.

Example: it is easy to observe in the figure the evolution of the battery of one device. In the current and static algorithm configuration, the evolution is lineal because it is always the same; in the dynamic algorithm it changes, therefore the configuration also changes and so, the connection of the device will be different, and then the consumption will be too.

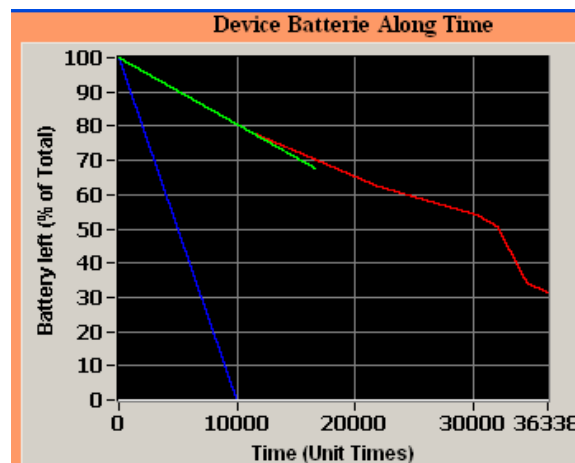
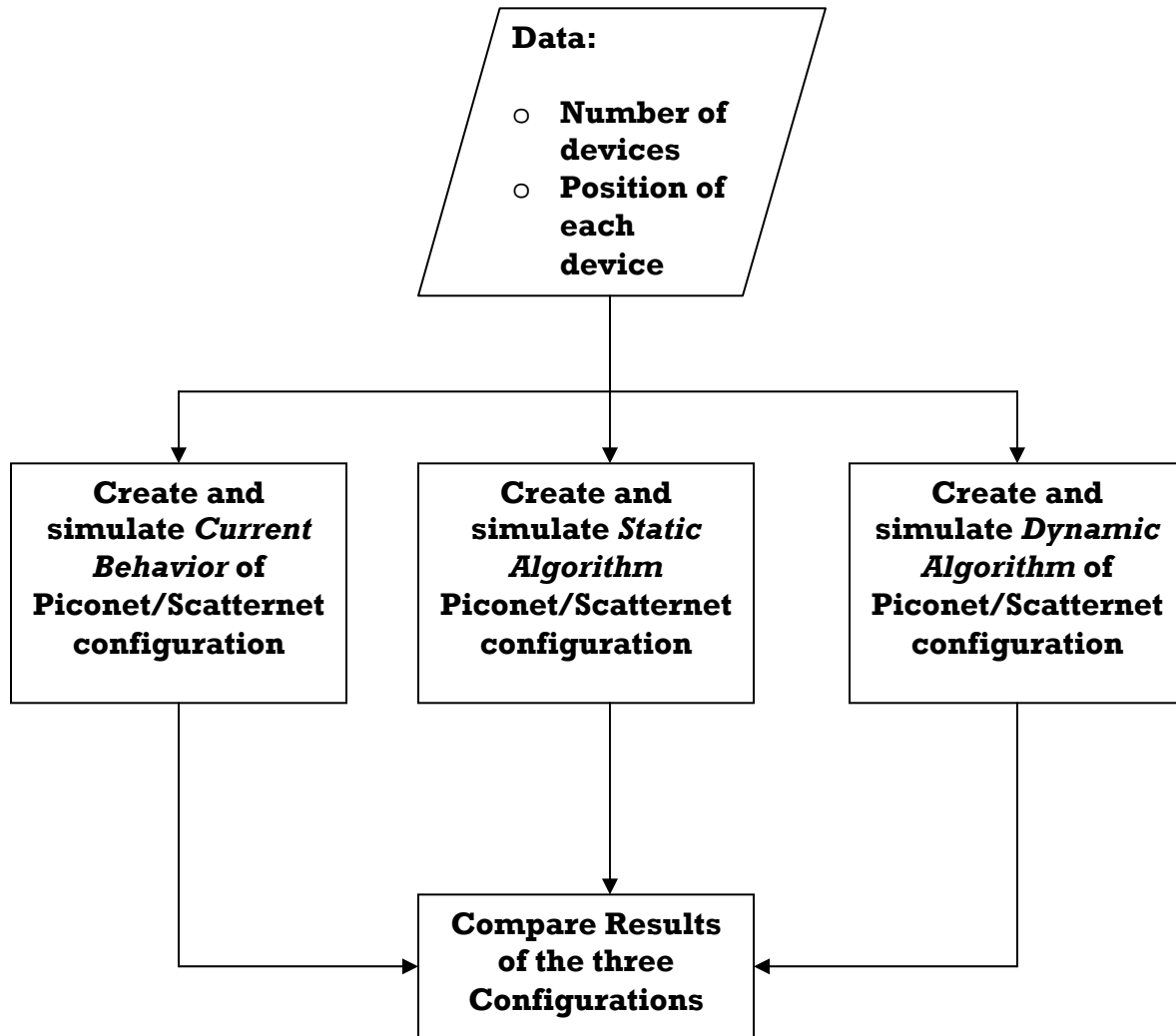


Figure 12: Battery evolution along time

➤ All this is summarized in this useful block diagram:



2.3.2 The algorithm

In the Bluetooth specification, there is no algorithm defined for scatternet creation. The main goal of this project is to develop one that succeeds in getting battery life optimization. This section is divided into some parts, thus it is easier to understand.

2.3.2.1 Considerations

- This algorithm has been designed to create a piconet/scatternet configuration to get battery life optimization. That means, that the algorithm just needs like data entry the position of each device in the space (coordinates x and y), and it will give the whole piconet/scatternet configuration: which the masters and which the slaves are, the way to connect the different piconets (if there are several), and some other useful data such as the increase of the life of the piconet/scatternet, or the evolution of the battery consumption of each device.



- The main goal of this algorithm is the battery life optimization. For this reason, it will create a configuration with the **lowest number of piconets**. The less number of piconets there are, the less number of masters and bridge devices there are, and so the life of the piconet/scatternet will be increased, because the number of critical devices is reduced as much as possible.

Critical devices: those devices that have higher battery consumption and then determine the life of the piconet/scatternet. Those critical devices will normally be the masters due to Bluetooth way to work: masters are always working, while slaves only work when their turn to work comes (slot). See Theoretical Introduction for more details.

example: If there are 7 devices, it will be made a configuration of just one piconet (if this is not possible, it would be with two or more); if there are 15 devices a scatternet made of two piconets will be tried...

- Although the algorithm needs the positions of the devices, this will not be necessary in the future when Hardware Experimenting will be done. It is only needed the coordinates of the devices to simulate a way of measuring the power that each device receives from the rest (One Bluetooth device is able to read the power that receives from another device with the RSSI parameter).
-

During this project, it was tried to make hardware experimenting, unfortunately there was no time enough to finish it, but the idea (that may be resumed in future projects) is as follows: to control and to communicate with each device using a microcontroller or PC, using HCI commands. With the HCI commands, the Bluetooth devices can be ordered whatever wished. The most important orders for this kind of project are:

- **HCI RESET:** this command is used to reset a Bluetooth device.
- **HCI READ BD ADDRESS:** command to read the BD ADDRESS of a device.
- **HCI INQUIRY:** with this command a device is told to look for discoverable devices. The Bluetooth device will indicate the devices found and their BD ADDRESS.
- **HCI CREATE CONNECTION:** once the inquiry has been made, it can be created a connection with the discovered devices. With this command the device is ordered to create a connection with one device with a determined BD ADDRESS.
- **HCI READ RSSI:** when there is a connection between two Bluetooth devices, it can be read how much power one device receives from the other. To do that, the HCI READ RSSI command is used just indicating the BD_ADDRESS of the wanted device.

This command is the most important one for this project, because it will be used to get all the information we need to run our algorithm.

- **HCI DISCONNECT:** this command is used to finish a connection between two Bluetooth devices.
- The only information that the algorithm needs to work, is for every device the received power from the rest (RSSI). But this is a simulating project, where there are no real Bluetooth devices, so it is not possible to read the RSSI. The solution is to use the position of each device (in the hardware experimenting this information will not exist), Thus the distance between each of pair of devices can be simulated using the *Pythagoras Formula*, and the received power from each one can be calculated using the *Friis Formula*.

FRIIS FORMULA

$$\text{Received Power} = 5 + (20 * \log[0.5]) - (20 * \log[\text{distance}])$$

As seen in the Friis Formula for Bluetooth, it is really easy (just the distance is needed) to calculate the power that one device receives from another, when there is a connection between them. Of course, the value obtained here is not going to be the same it would be obtained using the HCI READ RSSI command, because in fact, there will be a lot more factors that cannot be considered just using the Friis Formula. But this is not a problem, the algorithm will also work properly with the information from the HCI READ RSSI: **the quality of this algorithm is independent from the quality of the information.**

- As it happens in all the communication radio systems, the communication between two devices is only possible when the power that each device receives from the other is higher than a threshold value. If the received power is under this value, there is no way to communicate those devices. For this simulation, it has been considered that this value is -60 dBm. This means, that if one device receives from another one less than -60 dBm, the first one is not able to discover the second one, and so it is impossible to communicate them.

In this project, when the power that one device receives from a second one is higher than -60 dBm it will be said that the first device **can see** the second one; if it is lower than -60 dBm it will be said that the first device **cannot see** the second one.

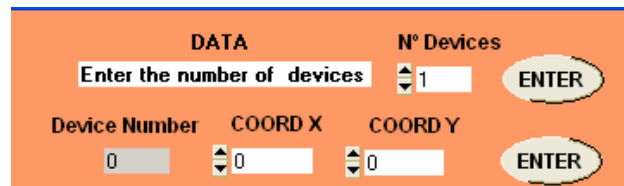
- In this model of simulation it has been considered the case where both, masters and slaves, have always something to transmit, and it is always the maximum. In real behavior, of course, this is not like this. For instance the masters hardly have something to transmit, and there will be slaves that will transmit much more than others.

Depending on the Bluetooth application, the masters will consume more or less, the slaves will consume more or less... This model of simulation works on the generic case, considering that masters and slaves have always something to transmit. Fortunately, if needed, it would be really easy to adapt the algorithm to a certain case, just changing some values. In this way, this is another advantage of this algorithm: **it can be adapted to any Bluetooth application.**

- For this algorithm, 5 sub-algorithms have been developed. All of them will be explained one by one. The idea is the same for all them (except for the random one), but some little points may be different. Therefore, they will be explained separately. These are the 5 sub-algorithms:
 - ✓ Random (Current) Algorithm Configuration
 - ✓ Static Algorithm for one piconet
 - ✓ Dynamic Algorithm for one piconet
 - ✓ Static Algorithm for scatternet
 - ✓ Dynamic Algorithm for scatternet

- All of the sub-algorithms begin working the same way. Therefore, these starting steps that are common for all of them, are explained just once:

I. Introduction of the positions of the Bluetooth devices: the coordinates (coordinate X and coordinate Y) of each device are inserted using a graphical user interface.



The screenshot shows a graphical user interface for data entry. It has an orange background. At the top, there are two labels: 'DATA' and 'N° Devices'. Below 'DATA' is a text input field containing 'Enter the number of devices'. To the right of this field is a spinner control showing the number '1'. Below 'N° Devices' is another spinner control showing '1'. To the right of these two controls is an 'ENTER' button. Below this section, there are three labels: 'Device Number', 'COORD X', and 'COORD Y'. Below 'Device Number' is a text input field containing '0'. Below 'COORD X' is a spinner control showing '0'. Below 'COORD Y' is another spinner control showing '0'. To the right of these three controls is another 'ENTER' button.

Figure 13: Data entry

II. Calculate the distance between each pair of devices: using the Pythagoras Formula. Considering two devices A and B, where A coordinates are A_Coord-X and A_Coord-Y, and B coordinates are B_Coord-X and B_Coord-Y, the distance between them can be calculated this way:

PITAGORAS FORMULA

$$\text{Distance} = \sqrt{A_Coord-X - B_Coord-X / ^2 + A_Coord-Y - B_Coord-Y / ^2}^{1/2}$$

III. Calculate the power that each device receives from the rest: just using the distance (calculated in the previous point) in the Friis Formula, the value of the power that one device receives from another is obtained. In this model of simulation it has not been considered some aspects that affect the real received power value, such as interferences, climatological conditions... Anyway, as it has been said before, the algorithm will work properly regardless of the quality of the information.

FRIIS FORMULA

$$\text{Received Power} = 5 + (20 * \log[0.5]) - (20 * \log[\text{distance}])$$

2.3.2.2 Static and Dynamic Algorithm

Two algorithms have been developed trying to solve the battery life problem: the first one is the Static Algorithm, which goal is to get the best piconet/scatternet configuration; the second one is the Dynamic Algorithm, and basically the way in which it works is linked to create several configurations using the Static Algorithm for each one.

- **Static Algorithm:** the main goal of this one is to find the best configuration for the piconet/scatternet: it will be chosen the configuration that gives the best battery life optimization. The algorithm will indicate which the masters are, which the slaves are and the way to connect the different piconets (if there are several ones). With the chosen configuration, it will be simulated the behavior along time, till one of the devices (one master) is with no more battery, so the piconet/scatternet is dead.

The simulating results are really good, because the life of the scatternet prolongs 25-70% from the current life. However, there is one problem that also happens in the current configurations: when the piconet/scatternet dies (one device with no battery), the masters are the ones that have their battery empty or almost empty, while the rest of the devices (slaves) have most of their battery remaining. That means that a lot of devices still have a lot of battery and the piconet/scatternet is not working anymore. Trying to solve this and to make a better use of the battery of all the devices, (that succeeds in increasing the life of the scatternet) the Dynamic Algorithm is developed.

Idea: the Bluetooth devices are configured in a piconet/scatternet in such way that there is a battery life optimization. However, the piconet/scatternet keeps on dying because of the masters: the masters spend most of their battery while the slaves keep most of it.

- **Dynamic Algorithm:** this algorithm is some kind of continuation of the static one. Furthermore, it is tried to make a better use of the battery of all the devices, and so to prolong the life of the scatternet.

The idea is using several configurations instead of using only one like it happens nowadays, or in the Static Algorithm. Each configuration will not be used till one device (normally a master) has no battery left, but it will be used till the battery of one master reaches a threshold value. At that point a new configuration will be performed. The dynamic algorithm can be summarized in two important points:

- **Each configuration will be valid while the battery left in the masters is higher than a threshold value.** This value will be automatically calculated by the algorithm, and the best value to obtain the longest life of the
-

piconet/scatternet fluctuates between 25 and 35% of the medium battery value of all the devices.

- When it is time to change the configuration (or if it is the first one), a new one will be created, using the static algorithm but with one restriction: **only the devices with a battery value higher than a threshold value will be able to be used as masters**. With this limitation, it is secured that the life of the configuration is not going to be too short. This is really important, because whenever a configuration is created, there is big battery consumption.

The threshold value, that a device needs to be master, fluctuates between 80 and 55% of the medium battery value of all the devices. Like in the other case, it is automatically calculated by the algorithm and the chosen value, in each case, is optimized to obtain the longest life of the piconet/scatternet.

After simulating with the dynamic algorithm, good results are really obtained such as: great battery spending of most devices (not only a few ones) and with that the life of the piconet/scatternet is increased in more or less about 300% of the beginning (current) value. With these results, just to conclude, it must be said that the idea of a dynamic algorithm is great and in the future, it will be a big field for researching.

idea: the dynamic algorithm is used to make a better use of the battery of all the devices. To do that, several configurations are used, changing them when the battery left in the masters is under a threshold value. Every configuration is created using the static algorithm, but with the limitation that only the devices with more battery than a threshold value can be chosen as masters.

2.3.2.3 Random (Current) Algorithm

➤ **Sequence for the Current Algorithm**

This random algorithm has been developed, looking for a way to simulate the real Bluetooth behavior, so the results of the algorithm could be compared with something real. The way this algorithm works is really simple:

- 1. Introduction of the positions of the Bluetooth devices.**
- 2. Calculate the distance between each pair of devices.**
- 3. Calculate the power that each device receives from the rest.**
- 4. Create a random configuration:** some devices will be selected randomly to work as masters and try to create a valid scatternet configuration; a valid piconet/scatternet configuration is that one that fulfils the following:
 - Every device which is not working as a master, must be working as a slave connected to a master.
 - The piconetes must not have more than 8 devices.
 - The piconetes must be properly connected.

Creating the random configuration, the next steps can be distinguished:

- I. Select randomly N masters:** at the beginning, N will be the lowest number possible to create the configuration, but if after some attempts a valid configuration has not been created, N will be increased in one unit: $N=N+1$, and try again.
- II. Check if it is possible to create a configuration with the N masters:** to do that, two conditions must be fulfilled:
 - 1) All the devices are discoverable:** this means that with the N masters, all the devices can be connected to one or more masters, and that the different piconets can be connected.

If this condition is fulfilled, the slaves will be assigned to the masters. It will be done this way:

- First, the devices that just *see* one master, will be connected to that master.
-

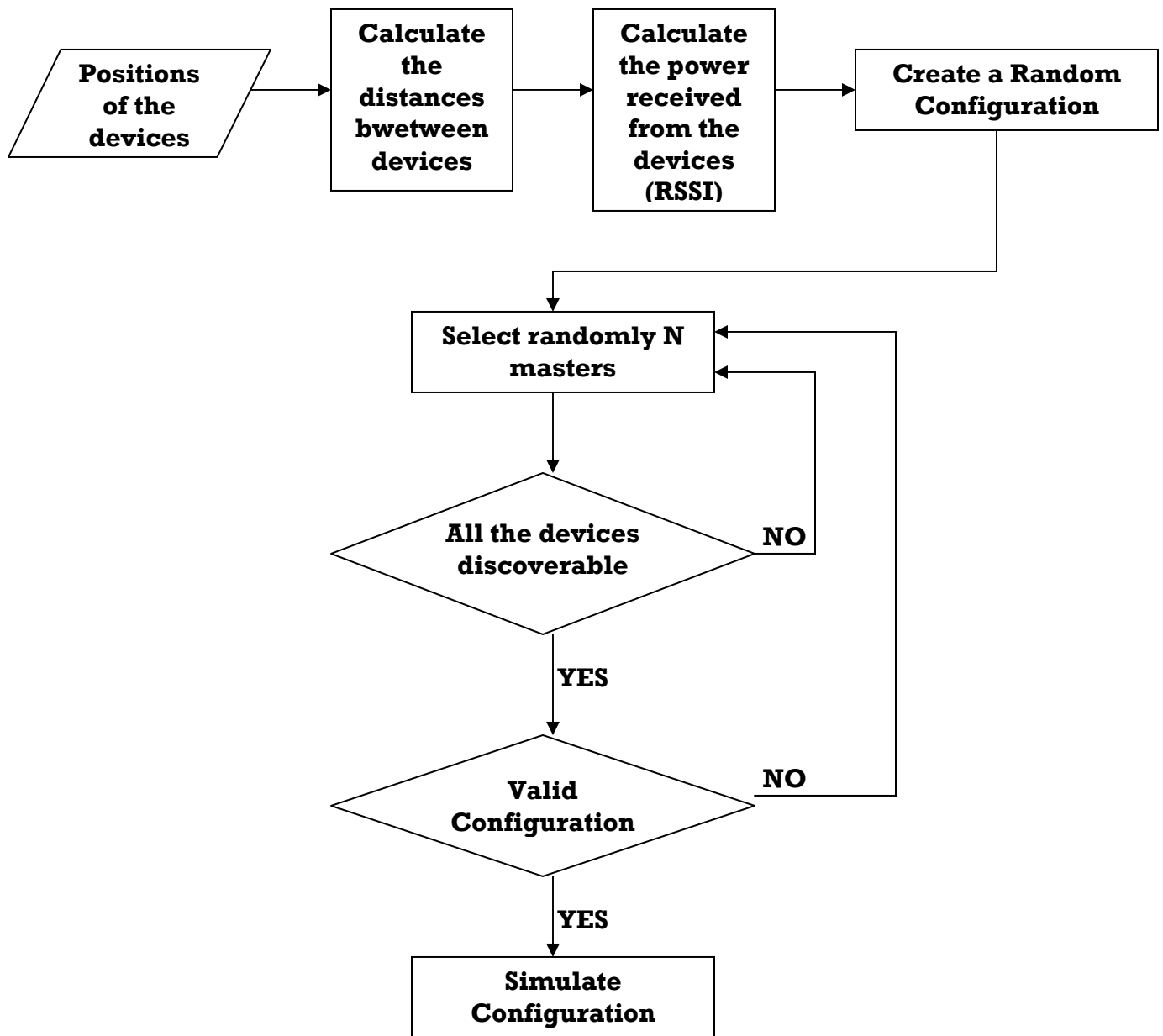
- Second, the rest of the devices will be randomly connected to one master that they can see.

2) The configuration created is a valid one: the configuration will be a valid one, if all the devices are connected to one piconet, if the piconets are connected properly between them, and if the number of devices of each piconet is not higher than 8. If the configuration is a valid one, the sequence jumps to point 5, and if it is not a valid one, it comes back to point 4.I to look for new masters and try a new configuration.

5. Simulation of the chosen configuration: the chosen configuration will be simulated along time, till it dies. This moment will happen when one master consumes all its battery. As it has been said several times, the masters' consumption, considering that masters and slaves always have something to transmit, is higher than the slaves' consumption, and so the masters spend faster their battery. With the simulation, it is obtained:

- Life of the configuration.
- Evolution of the consumption of the devices along time.

➤ **Block Diagram for the Current Algorithm**



2.3.2.4 Static Algorithm for one piconet

The algorithm developed for the case of a piconet, is a bit different from the one developed for a scatternet made of several piconets. Here, it will be explained the algorithm for one piconet in the static mode: it must be reminded that the static algorithm looks for the best configuration, in order to prolong as much as possible the life of the piconet. The chosen configuration will be simulated until one device is with no more battery and the piconet is then dead.

As said in the *Considerations Section*, the algorithm will create the configuration with the lowest number of piconets possible. So, this algorithm will be used as long as the number of devices is no higher than 8 (if it is not possible, it will be tried with 2, 3... piconets). The sequence for the algorithm is as follows:

➤ **Sequence for the Static Algorithm for One Piconet**

- 1. Introduction of the positions of the Bluetooth devices.**
- 2. Calculate the distance between each pair of devices.**
- 3. Calculate the power that each device receives from the rest.**
- 4. Check which devices could work as master:** one device can work as master if it *sees* all the devices. That means that it must receive more than -60 dBm from the rest of the devices. If it can see all the devices, it could work as a master because it can support communication with all of them and so create a suitable piconet.

If there were no devices that could work as master, it is not possible to work with just one piconet, and a scatternet made with two piconets will be tried either.

- 5. Choose the master between the candidates devices:** for every candidate device a *Candidate Value* is calculated, and the device with the highest value, will be chosen as master. Once the master is chosen, the remaining devices will be connected to the master working as slaves. Therefore, the piconet will be created.

The *Candidate Value* for one device is calculated like the addition of the power values that the device receives from the others. For instance, if there are 7 devices and device number 1 is a master candidate, its *Candidate Value* is the addition of the power received from the devices 2, 3, 4, 5, 6 and 7.

In this way, it is guaranteed that the configuration with the chosen master is the one that prolongs as much as possible the life of the piconet. For the candidate devices it can be expressed throughout a mathematical formula which device will be chosen as master:

MASTER OF A PICONET

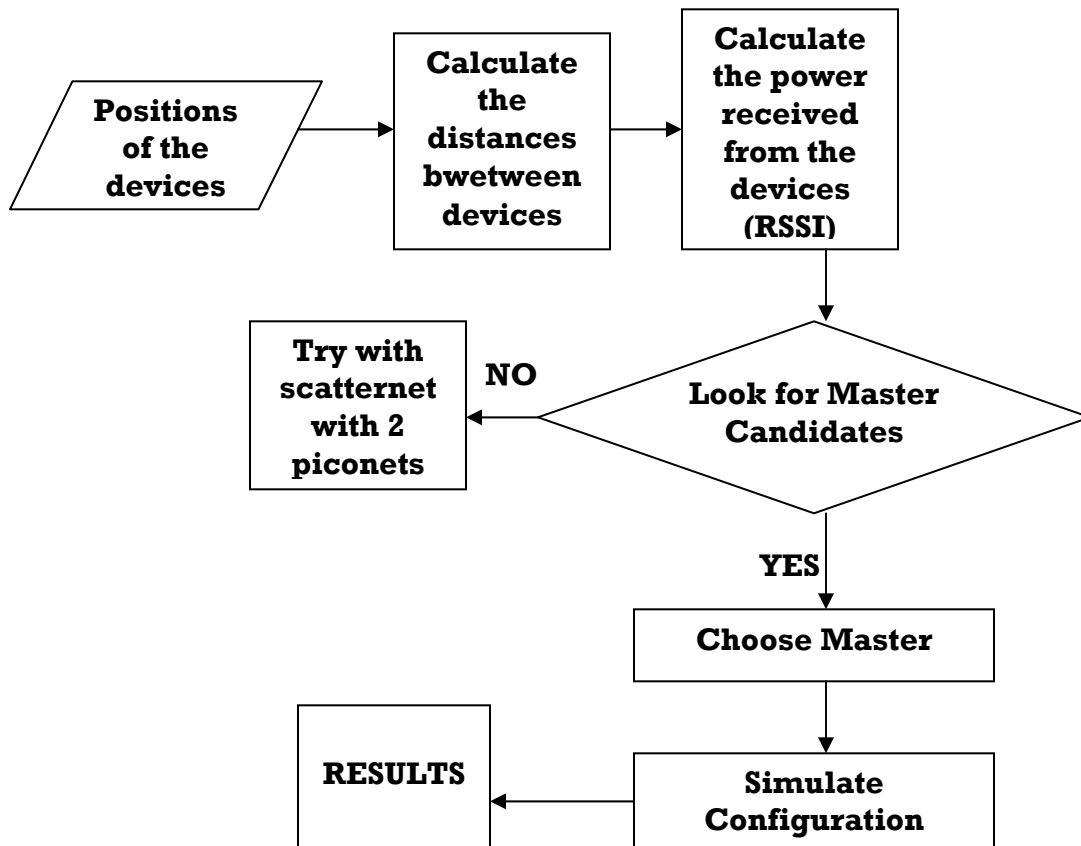
$$\text{Master} = \max [\sum(\text{RSSII})]$$

6. Simulation of the chosen configuration: the chosen configuration along time will be simulated, until the piconet dies. This moment will happen when the master consumes all its battery. As it has been said several times, the master consumption, considering that masters and slaves always have something to transmit, is higher than the slaves' consumption, and then the master spends faster its battery.

7. Analyze the results: with the simulation, the following results are obtained:

- Life of the configuration.
- Evolution of the consumption of the devices along time.

➤ **Block Diagram for the Static Algorithm for One Piconet**



2.3.2.5 Dynamic Algorithm for one piconet

The dynamic algorithm is born trying to make better use of the devices batteries, in order to prolong the life of the piconet. With the static algorithm, the master was the device which spent fastest its battery, while most of the slaves still had most of their battery remaining. The idea is to try to consume the battery of almost all the devices, to prolong the life of the configuration.

The dynamic algorithm will use the static algorithm several times, till one device has no battery left (the piconet will be dead). Every time a configuration is created with the static algorithm, two points must be considered:

- **Not all the devices can be used as master:** only those ones whose battery left is higher than a threshold value. After making a lot of attempts, it was concluded that the best value of this threshold value is the 50% of the medium amount of battery of all the devices, in order to prolong as much as possible the life of the piconet.
- **One configuration will work until:** the battery left in the master is lower than a threshold value (40% of the medium amount of battery of all the devices) or until one device is with no more battery. If the configuration finishes working for the first reason, a new configuration will be created and go on working; if it finishes for the second one, the piconet has died and the algorithm is over.

The threshold values used before: 50% of battery left to be able to be master, and 40% of battery left in the master to finish the configuration, have been settled after making a lot of attempts with all the possible values. With the 50%-40%, these goals are achieved:

- **The battery of almost all the devices have been spent, when the piconet dies.** That means that a better use of the battery of the devices has been made, and so the highest life for the piconet is obtained.
 - **The life of each piconet configuration is long enough to compensate for the battery consumption that is done in every configuration change:** whenever the configuration of the piconet is changed, a lot of battery is spent because all the connections between master and slaves have to be created, the master must synchronize all the slaves... This entire means, that the idea of the dynamic algorithm of changing the configuration is not as good as it seems. If a lot of changes are made, a lot of battery will be spent, and if the duration of each configuration is not too long, the change will not be worth enough. With the 50%-40% threshold values, the obtained configurations have quite long life and the number of changes is not too high. Therefore the dynamic algorithm becomes a really worth one.
-

➤ **Sequence for the Dynamic Algorithm for one piconet**

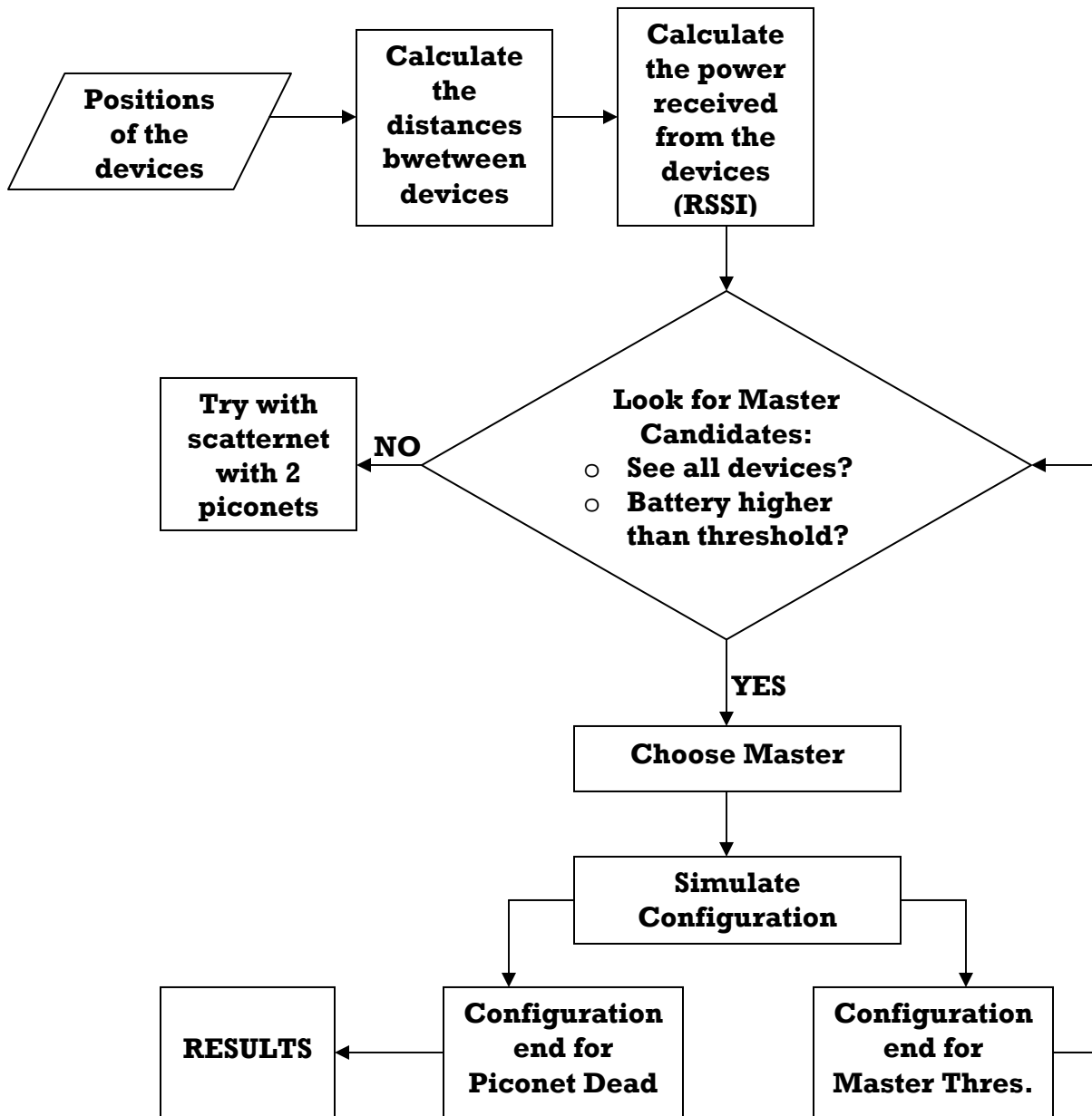
- 1. Introduction of the positions of the Bluetooth devices**
- 2. Calculate the distance between each pair of devices.**
- 3. Calculate the power that each device receives from the rest.**
- 4. Check which devices could work as master:** one device must fulfill two conditions to be able to work as master:
 - It must *see* all the devices, so all the devices can be connected to the piconet. This condition was the only one that a device had to fulfill to be a master candidate in the static algorithm. Now the devices must fulfill another one.
 - Its battery left must be higher than the 50% of the medium amount of battery of all the devices.
- 5. Choose the master between the candidates devices:** the way to choose the master is exactly the same one used in the static algorithm, calculating for every device the *Candidate Value* and choosing the device with the highest one.

<p style="text-align: center;">MASTER OF A PICONET</p>

<p style="text-align: center;">Master = max [$\sum(RSSI)$]</p>
--

- 6. Simulation of the chosen configuration:** the chosen configuration will be simulated along time. The simulation may finish for two reasons:
 - Because the battery left in the master of the piconet is lower than the threshold value (40% of the medium amount of battery of all the devices). In this case, this configuration is over but the piconet is still alive, and a new configuration will be created, coming back to point 4.
 - Because one of the devices has no battery left. In this case, the piconet has died and this one was the last configuration. The only thing left is to analyze the results.
 - 7. Analyze the results:** with the simulation, it is obtained:
 - Life of the configuration
 - Evolution of the consumption of the devices along time
-

➤ **Block Diagram for the Dynamic Algorithm for one piconet**



2.3.2.6 Static Algorithm for scatternet

The algorithm developed for a scatternet made of several piconets, is more or less the same one developed for just one piconet, but it is a bit more complex, because some other factors must be considered now:

- ✓ **The way to connect the piconets:** our goal is looking for the best way to communicate the different piconets, in order to reduce the battery consumption as much as possible. As it was said the *Theoretical Introduction* there are two ways to connect two piconets:

- **Connecting directly both masters of the piconets:** in this case, one master will work as master in one piconet and as a slave in the other one.

Example: in the figure, piconets A and B are connected this way

- **Using a bridge slave:** in this case a device will work as a slave in both piconets.

Example: in the figure, piconets A and C are connected this way

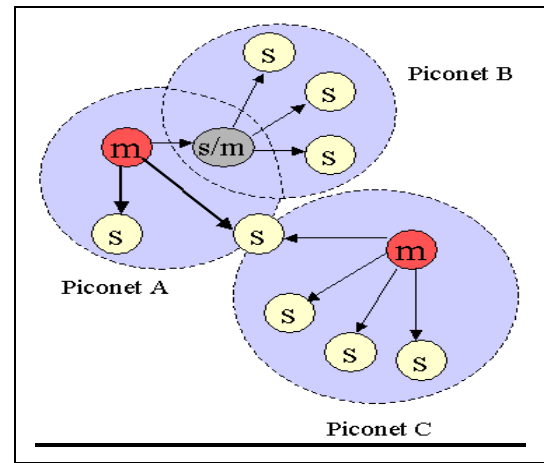


Figure 14: Scatternet

- ✓ **The devices can be connected to more than one master:** there will be devices that will be *seen* by more than one master. They will be connected to one or another master, in order to prolong as much as possible the life of the scatternet.

These two factors will complicate the search for the best scatternet configuration, and so the algorithm will be more complicated. Once the best configuration is obtained, it will be simulated along time till one device, one of the masters or a bridge slave is with no battery left and the scatternet is then dead.

This algorithm will be used in two cases:

- ✓ **When it is not possible to create a configuration of just one piconet:** when there are till 8 devices, a configuration of just one piconet is tried, but as it has been said, if it is not possible to do so, it will be tried to make a scatternet made of 2, 3, ... piconets, using this algorithm.
- ✓ **When there are more than 8 devices, it will be tried to make a scatternet with the lowest number of piconets possible:** thus, if there are till 15 devices, a scatternet of 2 piconets is tried, if there are till 22 devices a scatternet of 3 piconets, and so on.

➤ **Sequence for the Static Algorithm for Scatternet**

- 1. Introduction of the positions of the Bluetooth devices.**
- 2. Calculate the distance between each pair of devices.**
- 3. Calculate the power that each device receives from the rest.**
- 4. Try all the possible configurations:** when working with just one piconet, it was quite easy to make the configuration: at much there were 8 possible configurations (one for each device working as master), all of them were tried and the best one was chosen. Working with more than one piconet, it will be done the same: try all the configurations and choose the best one.
 - 4.1. Choose N devices to work as masters of N piconets:** N will be the lowest value possible. N devices will be chosen to work as masters of N piconets, and try a configuration with them. As said before, all the possible configurations will be tried: for instance if there are 15 devices it will be tried a configuration with the devices 1 and 2, 1 and 3, 1 and 4, ... working as masters.
 - 4.2. Try to build a configuration with the N masters:** to make a valid configuration, the N masters must fulfill these conditions:
 - **The rest of the devices must be *seen* by at least one of the masters:** if one device is not seen by any master, the configuration is not a valid one the algorithm will go to point 4.1 to look for new masters and try a new configuration.
 - **There must be at least one way to connect the N piconets:** it must be checked that it is possible to communicate the N piconets to create a scatternet. It is not necessary to connect directly the N piconets, but there must be one way to go from one piconet to another, although they are not connected directly by a bridge device.

If it is not possible to connect the N piconets, the algorithm will come back to point 4.1 to look for N new masters and try a new configuration; if it is possible, the N piconets will be connected in such way that the battery consumption is the lowest possible.

To connect the N piconets saving as much battery as possible, all the possibilities will also be tried: using the RSSI values calculated in the point 3, the battery consumption of all the possibilities of connecting the N piconets will be calculated, following two steps:

- First of all, for every couple of piconets it is essential to calculate the battery consumption, connecting them using a bridge slave (the best one is looked for), or just connecting both masters. It will be chosen the best option.
- Secondly, find the best way to connect the N piconets. As it is natural, it is not necessary to have a connection between each couple of piconets, but it is necessary to have a way, so it is possible to connect every couple of devices. The chosen option will be the one that saves more battery.
- **Connect the rest of the devices as slaves in the N piconets:** at this moment there are N devices working as masters, and N-1 devices working as bridges. These N-1 may be bridge-masters (a device which works as master in one piconet, and as a slave in another one) or bridge-slaves (devices that work as a slave in two piconets). Now it is time to assign the rest of the devices. They will work, just as slaves in one piconet. This will be done in three steps:

- I. **Devices only seen by one master:** the first devices to be assigned will be those ones that are only seen by one master. Obviously, if one device is just seen by one master, the device will work as a slave in that piconet, connected to that master.

After this point, the number of devices of each piconet is checked: if there is one piconet with more than 8 devices, the configuration is not a valid one and the algorithm will come back to point 4.1 to look for new masters and try another configuration.

- II. **Devices seen “better” by one master than by the rest:** at this moment, the devices that have not been assigned yet are devices that are not just seen by one master, but by several. At this point, the devices seen by one master with much more power (seen “better”) than by the rest, will assigned using the following mechanism:

- For each device, a *Difference Value* is calculated. This value is calculated as the subtraction of the highest RSSI received from one master, and the second highest RSSI received from another master.

$$\text{Difference Value} = \text{RSSI}_{1^{\circ} \text{HIGHEST}} - \text{RSSI}_{2^{\circ} \text{HIGHEST}}$$

- It is created a list with the devices, put in order in descending way by their *Difference Value*. In this list there will not be all the devices, but only those ones, which *Difference Value* is higher than
-

a threshold value. When the *Difference Value* of one device is lower than the threshold value; it means that this device is seen almost with the same power by two (or more) masters, and thus this is **not a critical device**: the life of the scatternet is hardly going to change, whether the device is in one piconet or the other.

- Finally the devices of the list are assigned. Every device will be connected to the master that *sees* it with the highest power, unless that piconet has already 8 devices. In that case, its *Difference Value* will be re-calculated and the device will be inserted again in the list or not (depending if the new *Difference Value* is higher than the threshold value or not). This will be done till the list is empty.

III. Devices seen by several masters more or less with the same power: at this moment, the devices that have not been assigned yet, are devices that are seen by at least two masters and the most important thing, they are seen by at least these two masters, with more or less the same power. This means that it is not going to be critical, whether they are connected to one master or to another.

This time, the masters will be the ones choosing a device, instead of being the devices the ones choosing a master (which is the way it happened in the I and II steps). It will be done in the following way:

- Every time a device has to be assigned, it is calculated for every master (except for the masters of the piconets which are already full) the *Spending Value* that will indicate the consumption of the master of each piconet. It is calculated this way:

$$\textit{Spending Value} = \frac{\sum_{i=1}^N (\textit{RSSI}_{\textit{SLAVES}})}{N}$$

Where $\sum(\textit{RSSI}_{\textit{SLAVES}})$ is the addition of the RSSI's of all the devices that are connected to that master (the slaves of that piconet) and N is the number of slaves of that piconet.

- It is easy to see that this *Spending Value* is independent from the number of devices of the piconet (as it is calculated dividing by N), which could be striking. But as it has been said in the *Theoretical Introduction*, if it is considered that the master and the slaves have always something to transmit, the consumption of the master is independent from the number of devices connected to it. To explain this, it will be reminded the way a piconet works: all the
-

communications are between the master and one slave. The master will communicate with each slave during a slot time, and then it will communicate with the next one. When all the slaves are done, the process will begin again and again, and so the master will always be working, independently from the number of slaves, and thus its consumption will not depend on this number, but it will depend on the medium amount of battery that it has to spend to communicate with their slaves.

- Once the *Spending Value* for every master is calculated, the one with the highest value will be the one choosing device, for being the one who has the lowest consumption. Now the question is the device to choose: it will be the device that the chosen master sees with the highest difference of power than any other master. For every free device, it will be calculated the *Difference Value 2*, in the following way:

$$\textit{Difference Value 2} = \textit{RSSI}_{\textit{CHOSEN MASTER}} - \textit{RSSI}_{\textit{I}^{\circ} \textit{HIGHEST}}$$

Where $\textit{RSSI}_{\textit{CHOSEN MASTER}}$ is the RSSI that the chosen master receives from the device, and $\textit{RSSI}_{\textit{I}^{\circ} \textit{HIGHEST}}$ is the highest RSSI that another master (which is not the chosen one) receives from the device.

- This point III will be repeated until all the devices are assigned. It could happen that it were not possible to assign all the devices following these steps. In such case, that configuration will not be a valid one, and the algorithm will come back to point 4.1 to try a new one.

- 5. Choose the best configuration between the candidates ones:** for every valid configuration, a *Configuration Value* will be calculated, and the one with the highest value will be the chosen one. This value is calculated according to this way:

$$\textit{Configuration Value} = \sum_{i=1}^M (\textit{Spending Value}_{\textit{MASTER } i})$$

Where M is the number of piconets, and *Spending Value* is the final value obtained in point 4.2.III. Moreover, it can be said that the masters of the chosen configuration will be those ones that fulfill this:

MASTERS OF A SCATTERNET

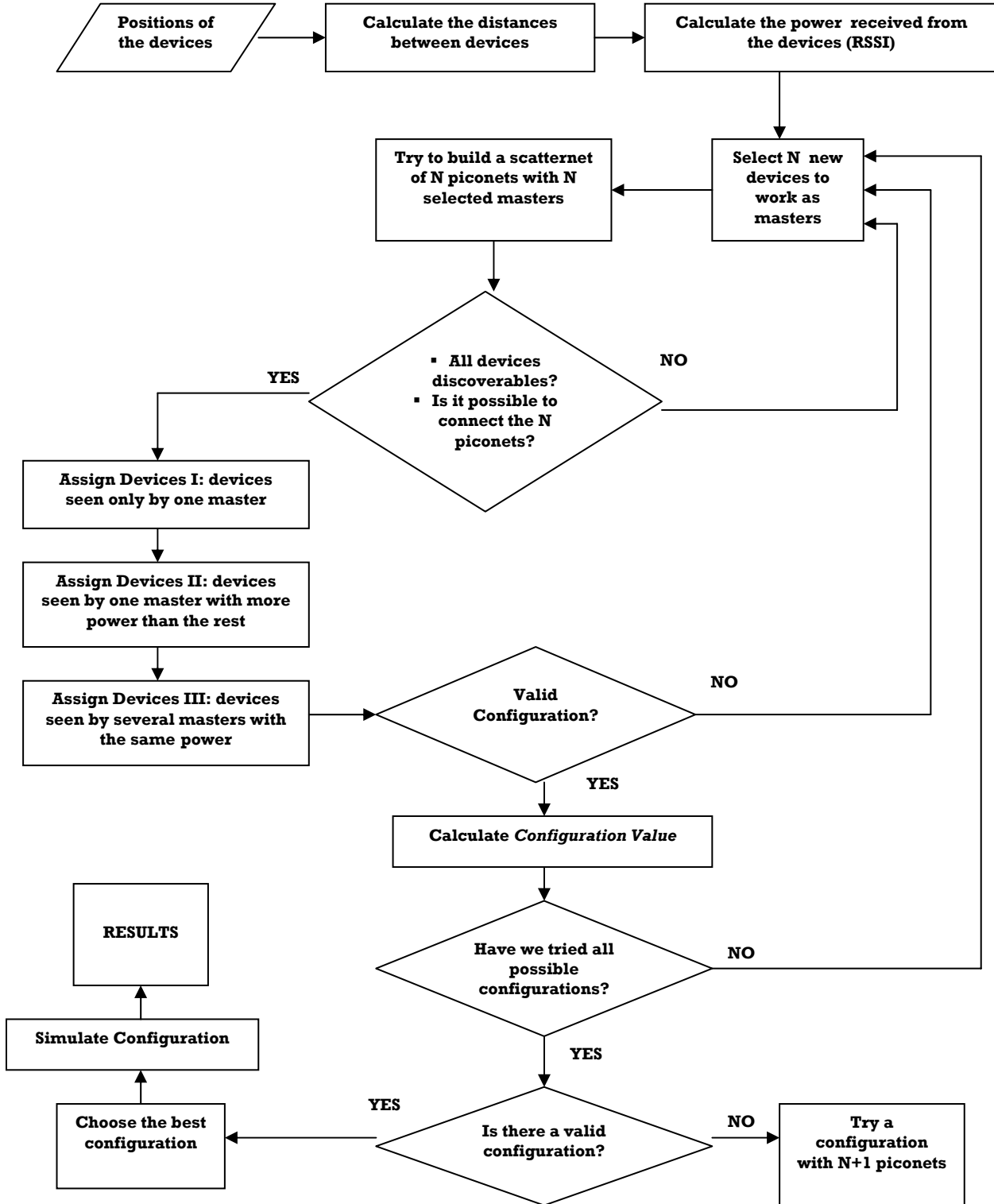
$$[M_1, M_2, \dots, M_M] = \max [Configuration Value + Bridges Value]$$

Where *Bridges Values* is calculated adding the power (RSSI) that each master receives from the device that is the bridge between its piconet and another one. This is a really important value to bear in mind, because connections between piconets are really important, more important than normal connections between a master and a slave: with the bridge connection, devices of two (or more) different piconets are connected, while with a normal connection, it is only possible to communicate two devices (the master and a slave). To praise highly this importance, the add of all these RSSI is multiplied by 2.

$$Bridges Value = 2 * \sum_{i=1}^M [(RSSI_{Br})_{MASTER i}]$$

- 6. Simulation of the chosen configuration:** the chosen configuration is simulated along time, till the scatternet dies. This moment happens when one master or a bridge slave spends all its battery.
- 7. Analyze the results:** with the simulation, the following results are obtained:
 - Life of the configuration
 - Evolution of the consumption of the devices along time

➤ **Block Diagram for the Static Algorithm for scatternet**



2.3.2.7 Dynamic Algorithm for scatternet

As it happened with the algorithm for one piconet, the dynamic algorithm for scatternet is born trying to make better use of the batteries of the devices to prolong the life of the scatternet. With the static algorithm, the masters were the devices that spent fastest their battery, while most of the slaves still had most of their battery remaining. The goal of this algorithm is to consume the battery of almost all the devices, in order to prolong the life of the configuration.

Instead of using just one configuration, the dynamic algorithm will use several scatternet configurations, in order to make a better use of the battery of all the devices and for prolonging the life of the scatternet. The end of one configuration can happen due to two reasons:

- **The battery left in one of the masters is lower than a threshold value:** in this case a new configuration will be created, where the devices with little battery left will not be allowed to work as masters, but the devices with quite battery left will be. Like this, the life of the scatternet is prolonged, spending the battery of almost all the devices, instead of spending the battery of a few devices that work as masters the whole life of the scatternet, as it happens nowadays or in our static algorithm.
- **One device has no battery left:** in this case, the end of the configuration coincides with the end of the scatternet. Checking the battery left in the devices, it is easy to see that most devices have spent or almost spent their battery, instead of been just a few ones, as it happened in the static case.

As it is known, the idea of the dynamic algorithm is to use several configurations for the scatternet. Every configuration will be created using the static algorithm, with the only change that not all the devices are allowed to be masters (only those ones whose battery value left, is higher than a threshold value). Every time a configuration is created with the static algorithm, two points must be considered:

- **Not all the devices can be used as master:** only those ones whose battery left is higher than a threshold value. The threshold value that a device needs to work as master, fluctuates between 80 and 55% of the medium battery value of all the devices. This value is automatically calculated by the algorithm and the chosen value in each case is optimized to get the longest life of the scatternet.
 - **One configuration will work until:** the battery left in one of the masters is lower than a threshold value. This value will be automatically calculated by the algorithm, and the best value to get the longest life of the scatternet, fluctuates between 25 and 35% of the medium battery value of all the devices.
-

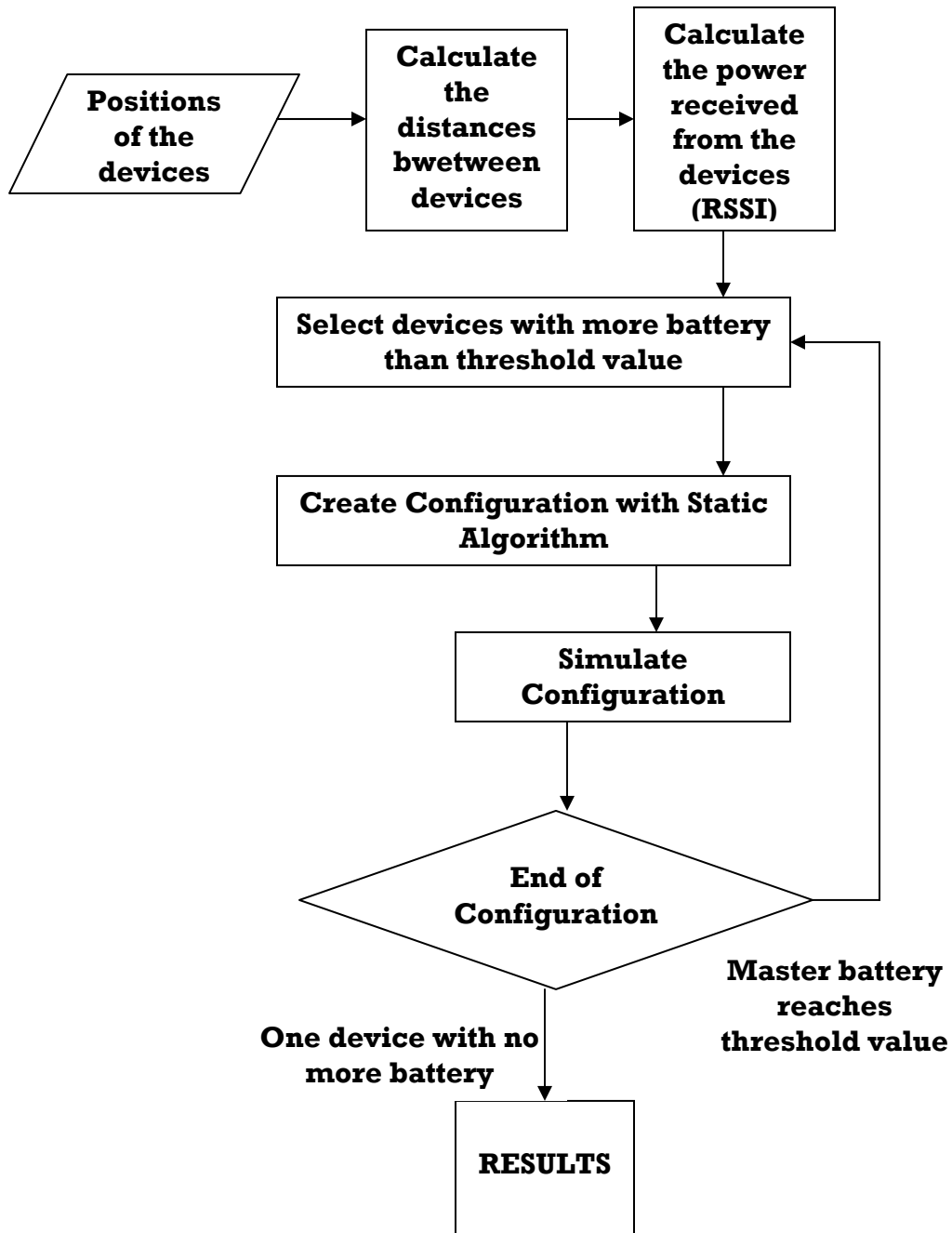
With these threshold values (that are automatically calculated and optimized by the algorithm), some important goals are achieved, the same way it happened in the dynamic algorithm for a piconet:

- **The battery of almost all the devices has been spent when the scatternet dies,** which means that it has been made a better use of the battery of the devices, and so a higher life of the scatternet is obtained.
- **The life of each configuration is long enough to compensate for the battery consumption that is done in every configuration change:** whenever the scatternet configuration is changed, a lot of battery is spent because all the connections among masters and slaves have to be created, the masters must synchronize their slaves... All this means, that the main idea of the dynamic algorithm, changing the configuration, is not as good as it seems. If a lot of changes are made, a lot of battery will be spent, and if the duration of each configuration is not too long, the change will not be worth enough. With the automatically calculated threshold values, configurations of quite long life are obtained, while the number of changes is not too high. Therefore, the dynamic algorithm becomes worthy really.

➤ **Sequence for the Static Algorithm for Scatternet**

- 1. Introduction of the positions of the Bluetooth devices.**
 - 2. Calculate the distance between each pair of devices.**
 - 3. Calculate the power that each device receives from the rest.**
 - 4. Create a new configuration:** in this point a new configuration is created. New configurations will be created till one device has no battery left and the scatternet is dead then.
 - 4.1. Select the devices that can be used as masters:** only the devices which battery left is higher than the threshold value, will be able to be masters.
 - 4.2. Create a configuration using the Static Algorithm for Scatternet:** this will be done just executing the static algorithm, with the only change that only the devices selected in point 4.1. as masters candidates, are able to be chosen as masters.
 - 4.3. Simulation along time of the chosen configuration:** the simulation will finish for one of these two reasons:
 - **The battery left in one master is lower than the threshold value:** in this case, the configuration is over but not the scatternet. A new configuration is created, coming back to point 4 then.
 - **One device has battery left:** in this case, both, configuration and scatternet, have finished. The algorithm will go to point 5 to analyze the results of the simulation.
 - 5. Analyze the results:** with the simulation, these results are obtained:
 - Life of the configuration
 - Evolution of the consumption of the devices along time
-

➤ **Block Diagram for the Dynamic Algorithm for scatternet**

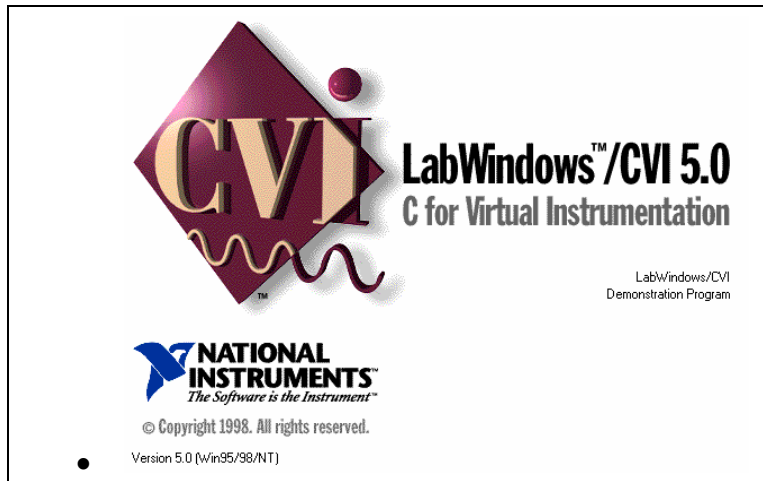


2.3.3 Software

2.3.3.1 Used Software Tools: Lab Windows

When the goal of the project was clear, another doubt came up: it was about what program was the best to build this model of simulation. Several options were thought about: Matlab, a simple C compiler... but finally Lab Windows was the chosen one because it is the perfect tool for this project. The reasons for this choice are the following ones:

- The algorithm needs a high level programming language, because it works with a lot of variables and loops. Nowadays, C language is known as one of the most powerful languages, and a lot of people who work in engineering are used to it too, so definitely, it comes to be the perfect language to develop this project. It could have been used another suitable option, such as Matlab or Pascal, but C language is not the only advantage of using Lab Windows.
- Thinking about the model of simulation, one really important matter is that a **graphical user interface** is needed, in order to have an easy way to introduce the coordinates of the devices, and an easy way to print graphs and charts, to show the results and conclusions. Lab Windows also fulfills this condition.
- Thinking about the future, next step that must be done is the **Hardware Experimenting**: put in practice the results of the algorithm and check how it works with a real Bluetooth behavior. So, it is really important to be able to control Bluetooth devices from one computer or micro-controller. With Lab Windows it is possible to **connect and control Bluetooth devices** by the parallel port, send and receive HCI-Commands, and so control them. In fact, it has also been developed a program, where it can be controlled till two Bluetooth devices, and send them some HCI-Commands, then it is possible to order an inquiry, to read the RSSI, to connect to one specific device... This part will be explained later, in the Future Steps section.



Lab Windows definitely fulfills these three conditions, and also a fourth one, which is our **experience** with this program: in Mälardalens Högskola; most of the projects that work with Bluetooth have been developed with this useful tool, and by myself too, I used to work with it, because it is one of the most used software tools at my home university (Alcalá de Henares).

Short Intro to Lab Windows:

Virtual instrumentation used to be rare in the test and measurement and industrial automation industries. Today, it has freed many scientists and engineers from the limitations of traditional instruments. Now more than ever, scientists and engineers recognize the benefits of integrating hardware and software components with PCs and workstations to create customized instrumentation solutions.

For traditional programmers who develop virtual instrumentation, LabWindows/CVI has become the most popular software development environment. Using this development tool, anyone with an elementary understanding of a text-based programming language can build virtual instruments. LabWindows/CVI combines the power of ANSI C and the flexibility of Microsoft Windows with easy-to-use tools for building virtual instrumentation systems.

Programmers can use the unique interactive code generation utilities in LabWindows/CVI to unleash power of ANSI C and speed development in the following areas.

- Creating and controlling graphical user interfaces under Windows
- Controlling instruments and plug-in data acquisition hardware from you PC
- Developing and debugging ANSI C programs for instrumentation

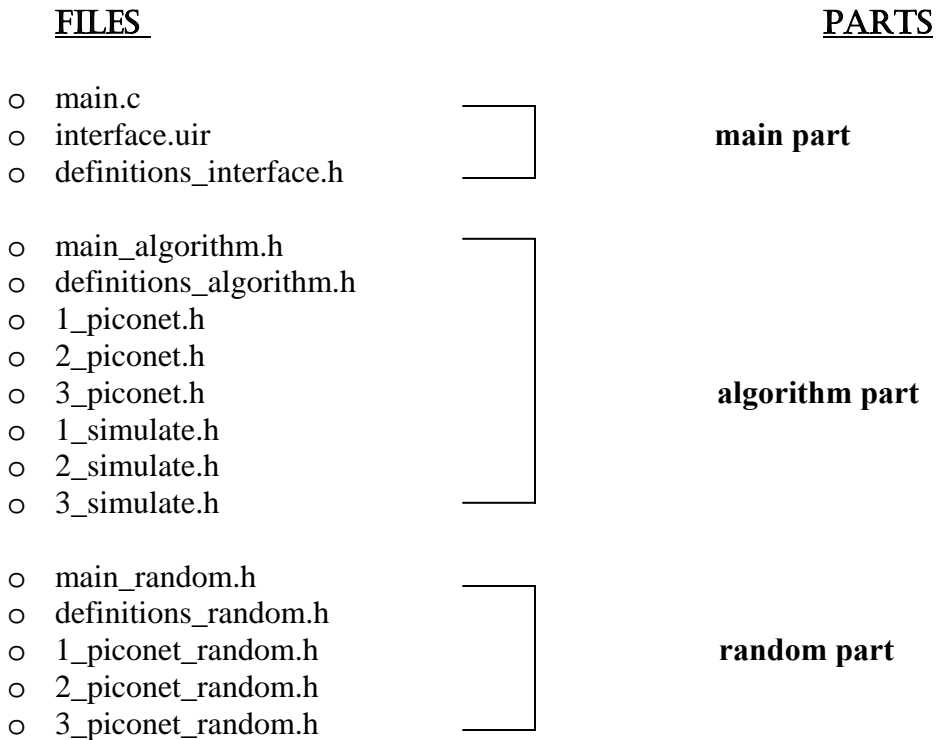


2.3.3.2 The program

Finally, in this section it is going to be explained the program developed to simulate the algorithm. It has been done in C language, using LabWindows. With this program the objectives explained in the previous sections are achieved:

- Create a current (random algorithm) to simulate nowadays Bluetooth behavior
- Create a static/dynamic algorithm for a piconet
- Create a static/dynamic algorithm for a scatternet made of a maximum of 3 piconets.
- Create a graphical user interface that provides an easy way to both, to introduce into the algorithm the Bluetooth devices wanted, and an easy way to control and to understand the results of the different algorithms.

The program has been written in a really smart way: the code is quite simple to understand, using basic C functions and it has been written in such way that can be easily modified. The program is divided into 16 files, but functionally it is divided into three main parts, where each one of them has a different task in the whole algorithm. The 16 files and the three functional parts are connected in this way:



Following, each functional part is explained, analyzing the files and source code of each one.

2.3.3.2.1 Main Part

❖ **Task**

This part is something like the brain of the program, the one that controls the other two parts, and the whole program in general. The *Algorithm* and *random* parts work creating and simulating the current, static and dynamic configurations, while this *main part* works coordinating everything, facilitating the data entry and showing the results of all the simulations in a suitable way. All these functions can be summarized just in the two following points:

- **Create a graphical user interface:** the interface gives the user the chance to do the data entry (number and coordinates of the devices) and the analysis of the results (configurations, life of each configuration and evolution of the battery consumption) in an easy way to understand.
- **Control the rest of the program to execute the algorithm:** it calls the necessary functions (functions that are in the other two main parts: algorithm and random) to execute the program, and create and simulate the three configurations (random, static and dynamic). It also takes the results and shows them in an easy way in the interface.

❖ **Files**

This part is built with 3 files:

- main.c
- interface.uir
- definitions_interface.h

▪ **main.c**

This file, as its name recalls, is the main one of the *main part* and also of the whole program. When the program is executed, this is the first file executed, so the best way to follow and study the source code, is just reading this file, and from here, the rest of the files will be called. From this file, it must be emphasized that:

- The file begins with some *#include*, to include the rest of the files of the program, like this all the functions of all the files can be called from this main file.

- Basically, there is just a function that opens the graphical user interface (this interface is in the file *interface.uir*), and the rest of the functions are *Callbacks Functions* used to control all the buttons that can be found in the interface: there are *Callback Functions* to read the number of devices, to read the coordinates of each device, to see the configuration created with the random algorithm, with the static, with the dynamic...

- The point where the other two parts (random and algorithm) are called, to create and to simulate the random, static and dynamic configurations, is located in the Callback function used to read the coordinates of the devices (Callback ReadCoordinates Function): when the coordinate Y of the last device is read, the function *main_2* is called. This function will create and simulate the static and dynamic configurations. After that, the function *main_random* is called and the random configuration is created and simulated.

- The results of the simulations will be visible, as soon as the *main_random* function is finished. Just after the code lines, where the functions *main_2* and *main_random* were called, it can be found in the lines that make the results visible in the interface. These lines show, where it can be seen how after reading the coordinates of the devices, the functions *main_2* and *main_random* are called and then it proceeds to show the results:

```
//We read the coordinates of the device and keep it coord_x and coord_y
GetCtrlVal (1, PANEL_NUMERIC_4_COORD_X, &coord_x[read_device-1]);
GetCtrlVal (1, PANEL_NUMERIC_5_COORD_Y, &coord_y[read_device-1]);

//We reset the numeric boxes where we write the coordinates
SetCtrlVal (1, PANEL_NUMERIC_4_COORD_X, 0);
SetCtrlVal (1, PANEL_NUMERIC_5_COORD_Y, 0);

//We have introduced the coordinates of one device, so we
//increase the number of devices read (read_device)
read_device++;

//When we have written the coordinates of all the devices we run
//our algorithm. First we run our algorithm (main_2), and then
//the random algorithm. This will be run so many time as necessary
//so we increase the life time of the scatternet at least in a 15%
if(read_device>num_dev)
{
    //We run our algorithm and print the results in the graphs
    main_2();

    do
    {
        main_random();
    }
```

```

}
while(life_random>(0.85*life_no_change));

//We print the evolution of the batteries of each device with time.We
//will print three graphs: the normal behavior, with algorithm and
//with algorithm and change of configuration
DeleteGraphPlot (1, PANEL_GRAPH, -1, VAL_IMMEDIATE_DRAW);
PlotY (1, PANEL_GRAPH, array_graph[0], life, VAL_DOUBLE,
VAL_THIN_LINE,VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);
PlotY (1, PANEL_GRAPH, array_graph_random[0], life_random, VAL_DOUBLE,
VAL_THIN_LINE,VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_BLUE);
PlotY (1, PANEL_GRAPH, array_graph_no_change[0], life_no_change,
VAL_DOUBLE, VAL_THIN_LINE,VAL_EMPTY_SQUARE, VAL_SOLID, 1,
VAL_GREEN);
PlotY (1, PANEL_GRAPH, array_graph_0, life, VAL_DOUBLE,
VAL_THIN_LINE,VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_WHITE);
SetCtrlVal (panelHandle, PANEL_NUMERIC ,1);

```

- The *Callback function Salir* is called when the Exit button is pressed. This function will quit the user interface and finish the execution program.

▪ **interface.uir**

In this file, it is created the graphical user interface. Buttons, number boxes, text boxes, charts,... are used for it. The user interface is open at the beginning of the execution (in the first lines of the main.c), and is closed when the Exit button is pressed. Following, the lines where the interface is open are shown:

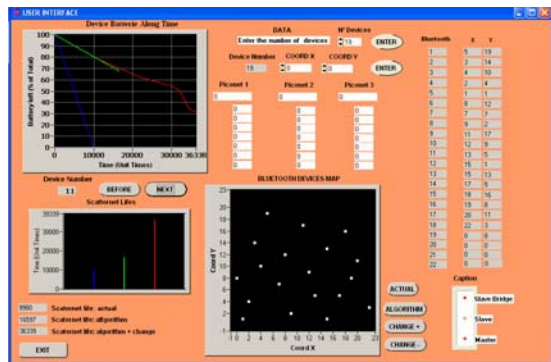


Figure 14: Graphical User Interface

```

int main (int argc, char *argv[])
{
    if (InitCVRTE (0, argv, 0) == 0)
        return -1; /* out of memory */
    if ((panelHandle = LoadPanel (0, "interface.uir", PANEL)) < 0)
        return -1;
    DisplayPanel (panelHandle);
    RunUserInterface ();
    DiscardPanel (panelHandle);
    return 0;
}

```

}

▪ **definitions_interface.h**

In this file, it can be found the definitions of all the items of the interface: the panel, buttons, text messages, number boxes, charts..., and the definitions of the Callback Functions that every Hot item has in. Some code lines are shown:

```
#define PANEL_GRAPH 111
#define PANEL_GRAPH_2 112
#define PANEL_GRAPH_3 113
#define PANEL_TEXTMSG_2 114
#define PANEL_TEXTMSG 115
#define PANEL_TEXTMSG_3 116

/* Menu Bars, Menus, and Menu Items: */

/* (no menu bars in the resource file) */

/* Callback Prototypes: */

int CVICALLBACK before(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int CVICALLBACK next(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int CVICALLBACK print_alg_change_before(int panel, int control, int
event, void *callbackData, int eventData1, int eventData2);
```


2.3.3.2.2 Algorithm Part

❖ Task

As it has been said before, the *main part* is dedicated to coordinate the other two parts and does not do anything related to the creation or simulation of the configurations, which is the task for the *algorithm* and *random* parts. While the random part is dedicated to create and simulate the random configuration, the algorithm one is dedicated to create and simulate the static and dynamic configurations. Its main functions can be summarized as follows:

- **Calculate the distances between devices:** using the information obtained in the data entry, and the Pythagoras Formula, the distance between every couple of devices is calculated.
- **Calculate the power (RSSI) that each device receives from the rest:** this value is easily calculated using the distance between each couple of devices, in the Friis Formula.
- **Create the static and dynamic configurations:** the creation is made following the steps explained in *The Algorithm* section.
- **Simulate the static and dynamic configurations:** with the simulation, these results are obtained: life of each configuration, battery consumption along time and the different configurations.

❖ Files

This part is built with 8 files, all of them *include* (*.h) files, that were included in the *main.c* in the first lines of this file. These are the files:

- main_algorithm.h
 - definitions_algorithm.h
 - 1_piconet.h
 - 2_piconet.h
 - 3_piconet.h
 - 1_simulate.h
 - 2_simulate.h
 - 3_simulate.h
-

▪ **main algorithm.h**

This file is the main one of this part, and it will control the creation and simulation of the static and dynamic configurations, calling the functions situated in the other files when necessary. Three parts are distinguished:

- **Information processing:** using the information of the data entry, it calculates the distances between devices, and the power (RSSI) that each device receives from the rest. Following, some line codes where the information processing happens:

```
/******FUNCION MAIN 2******/
//Main function where we will calculate the distances between
//each couple of devices, the received power by each device
//from the rest, the transmitted power by one device to the
//others, and using all these results we will run our
//algorithm and get the scatternet configuration. Finally we
//will simulate it and get the lifetime and the evolution of
//the batteries along time

void main_2 ()
{
    int i,j,p,k;

    calc_distances(); //We calculate the distance
    calc_rx_power(); //We calculate the rx power
    calc_tx_power(); //We estimate the tx power
    reset_1();       //We reset and initialize
    run_alg();       //We run our algorithm
}
```

- **Creation of the configurations:** as explained before, this algorithm has been designed to create scatternets using the lowest number of piconets possible, and has been designed to work with a maximum number of devices of 22, like this, with a maximum number of 3 piconets by scatternet.

To create the configurations, the files 1_piconet.h, 2_piconet.h and 3_piconet.h are used. The criterion used to call the functions of one or another file is the number of devices the algorithm is working with: if there are till 8 devices (1 piconet) , the 1_piconet.h file is used, if there are among 8 and 15 (2 piconets), it is used the 2_piconet.h, and, finally if there are among 16 and 22 (3 piconets), the 3_piconet.h is used. Following the code lines where the functions 1_piconet, 2_piconet and 3_piconet (situated in the same name files) are called to create the configurations:

```
//We begin to form the scatternet configuration

if (num_dev<=8)          //If we have less than 8 devices we
                        //will try just one piconet
{
    while (dead==0)
    {
        for(i=0;i<num_dev;i++)
            if(batterie[i]<=0)
                dead=1;
        if(dead==0)
            piconet_1();
    }
}

//If we have 9-15 devices, we will try two piconets

else if (num_dev<16 && num_dev>8)
{
    do
    {
        for(i=0;i<num_dev;i++)
            if(batterie[i]<=0)
                dead_2=1;
        if(dead_2==0)
            piconet_2();
    }
    while ((dead==0)&&(dead_2==0));
}

else if (num_dev<23 && num_dev>15)
//We will try three piconets
{
    do
    {
        for(i=0;i<num_dev;i++)
            if(batterie[i]<=0)
                dead_2=1;
        if(dead_2==0)
            piconet_3();
    }
    while ((dead==0)&&(dead_2==0));
}
}
```

- **Simulation of the configurations:** to simulate the configurations, the functions situated in the 1_simulate.h, 2_simulate.h and 3_simulate.h are called. With the 1_simulate.h, the simulation of all the configurations of one piconet are done, with the 2_simulate.h the simulation of the configurations of
-

2 piconets, and with the 3_simulate.h the simulation of the configurations of 3 piconets.

- **definitions algorithm.h**

In this file, it just can be found the definitions of the global variables and functions that are used in the *algorithm part*. Following, as an example, some lines from this file are shown:

```
void main_2(void);
void calc_distances(void);
void calc_rx_power(void);
void calc_tx_power(void);
void reset_1(void);
void reset_2(void);
void reset_3(void);
void run_alg(void);
void simulate_alg(void);
void piconet_1(void);

int master[5];
int final[40],final_2[40];
double min;
double batterie[40],connect[40][40];
int life,life_no_change;
int dead,dead_2;
int pic_1[20],pic_2[20], pic_3[20],pic_4[20];
```

- **1 piconet.h, 2 piconet.h, 3 piconet.h**

These files contain the functions used to create the static and dynamic configurations. The way they work was explained in the *algorithm* section, trying all the different configurations and choosing the best one.

These configurations will be shown in the *Bluetooth Devices Map* and in the *Piconets Members Part* of the interface.

- **1 simulate.h, 2 simulate.h, 3 simulate.h**

Finally, these files are called in the *main_algorithm.h* to simulate the static and dynamic configurations created in the 1_piconet.h, 2_piconet.h and 3_piconet.h. The simulation will be done in order to get:

- life of each configuration.
- evolution along time of the battery consumption of each device, in every configuration.

All these results obtained in the simulations will be returned to the *main part*, and this will show them in the interface.

2.3.3.2.3 Random Part

❖ Task

This part is really close to the *Algorithm Part*: it is also controlled by the *main part* and it is dedicated to create and simulate configurations too; the difference is that it will create and simulate the random configuration, instead of the static and dynamic ones created and simulated in the *Algorithm Part*. The main functions of this part are summarized in the following points:

- **Creation of the random configuration:** the creation will be made following the steps explained in *The Algorithm* section.
- **Simulation of the random configurations:** with the simulation, the following results are obtained: life of the configuration, battery consumption along time and the own configuration.

The information about distances between devices, and the power received from the rest of devices (RSSI) will be taken from the *Algorithm Part*.

❖ Files

This part is built with 5 files, all of them *include* (*.h) files, that were included in the *main.c* in the first lines of this file. The structure is the same as the *Algorithm Part*: one main file to control the rest (*main_random.h*), a file with the definitions of the variables and functions used (*definitions_random.h*) and 3 files to create and to simulate the random configuration (*1_piconet_random.h*, *2_piconet_random.h* and *3_piconet_random.h*). The only difference is that, in this case it is used the same file to create and to simulate the configurations, while in the *Algorithm Part*, two separated files are used for each task.

The files of this part are:

- *main_random.h*
 - *definitions_random.h*
 - *1_piconet_random.h*
 - *2_piconet_random.h*
 - *3_piconet_random.h*
-

▪ **main_random.h**

This file is the main one of this part, and it will control the creation and simulation of the random configuration, calling the functions situated in the other files when necessary. Depending on the number of devices, it will call one or another function, to create (and also simulate) a random configuration with the lowest number of piconets possible:

- **If the number of devices is no higher than 8:** it calls the *1_piconet_random* function.
- **If the number of devices is between 9 and 15:** it calls the *2_piconet_random* function.
- **If the number of devices is between 16 and 22:** it calls the *3_piconet_random* function.

Now it will be shown some code lines where it can be seen, how depending on the number of devices, one function or another is called:

```
//We call a function depending on the number of devices
do
{
    if (num_dev<=8)           //If we have less than 8 devices we will
        piconet_1_random();//try just one piconet

    else if (num_dev<16 && num_dev>8) //If we have 9-15 devices, we
        piconet_2_random();         //will try two piconets

    else if (num_dev<23 && num_dev>15) //We will try three piconets
        piconet_3_random();

}
while (random_ready==0);
```

▪ **definitions_random.h**

In this file, it just can be found the definitions of the global variables and functions that are used in the *random part*. Following, as an example, some lines from this file are shown:

```
void main_random(void);
void reset_random(void);
void piconet_1_random(void);
void piconet_2_random(void);
void piconet_3_random(void);
```

```
void masters_2_random(void);
void masters_3_random(void);
int coord_x_bridge_rand_1_2[8], coord_y_bridge_rand_1_2[8];
int coord_x_bridge_rand_1_3[8], coord_y_bridge_rand_1_3[8];
int coord_x_bridge_rand_2_3[8], coord_y_bridge_rand_2_3[8];
int bridge_rand_1_2,bridge_rand_1_3,bridge_rand_2_3;
```

- **1 piconet random.h, 2 piconet random.h,**
3 piconet random.h

These files are used to create and simulate a random configuration of 1, 2 or 3 piconets. The way they work is the same one explained in the *Algorithm Section*. That is:

- The creation of the configuration is done randomly.
- The simulation along time is done to get the same information that was taken for the static and dynamic algorithm, and so it can be compared and see the improvements:
 - ✓ Life of the configuration
 - ✓ Evolution of the battery consumption along time.

2.4. RESULTS

In this project, it has been developed an algorithm to create the configuration of a piconet/scatternet. The main goal was the battery life optimization, in order to prolong the life of the piconet/scatternet. In this section, it will be shown and explained the results obtained from the simulation of the algorithm, which are basically three:

- **Prolongation of the life** of the piconet/scatternet.
- **Optimization of the consumption** of the Bluetooth devices.
- **Optimization of the piconet/scatternet configurations.**

For each point, it has been done an explanation of the obtained results, using for that some drawings, charts... It has also been used the example used during the whole project, to show and analyze the results obtained in that case. Our example was the one with 19 Bluetooth devices, distributed in the following way:

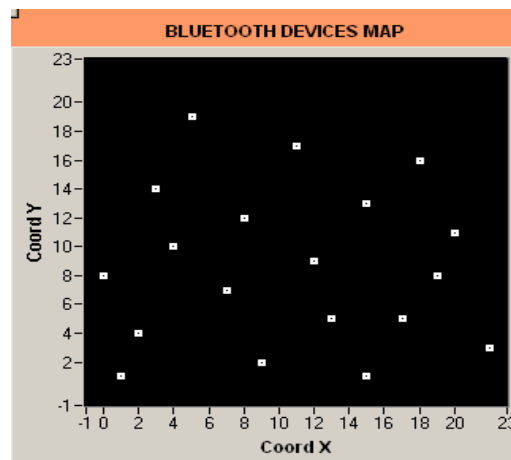


Figure 15: Initial map

2.4.1. Prolongation of the life of the piconet/scatternet

This section must be started, reminding the definition of *Life of the Piconet/Scatternet*: it is the time it takes since the piconet/scatternet begins working, till one device has no battery left. Nowadays this term is really important, because the masters are normally the devices that spend fastest their battery, and thus they are the first ones stopping working. That means, that the slaves of that piconet, even if their batteries are almost full cannot communicate with any other device of that or another piconet. That is why it is so important to increase the value of *the life of the piconet/scatternet* as much as possible.

As explained before, three different configurations have been developed:

- **Random (Current) configuration:** to simulate present Bluetooth behavior.
- **Static configuration and Dynamic configurations:** where the algorithm is put into practice.

It has been considered that the life of the random configuration, is the value used as reference to compare the results from the algorithm with. Related to this, it must be said that:

- With the **Static Algorithm**, it is obtained an increase of the life of the configuration that fluctuates between **25% and 75%** of the random configuration, while with the **Dynamic Algorithm**, it is obtained an increase that fluctuates between **200% and 300%**.

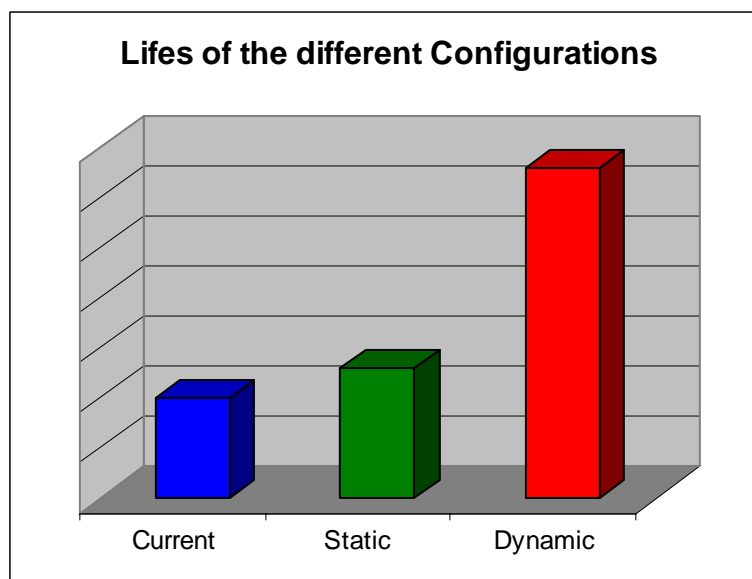


Figure 16: Lives of the different configurations

- The fluctuation of the results depends on several factors:
 - **The random algorithm:** it is been compared the life of the static and dynamic configurations, to the value of a random configuration. That means, that there will be better *random configurations* than others, and so the life of each configuration will be different.
 - **The number of devices:** when the number of devices increases, the results of the algorithm improve, while when the number of devices is lower, the results are not so spectacular. The reason is that when there are few devices, there are also very few configurations possible and so it is easy to create randomly a good configuration; when the number of devices increases, there are many possible configurations and it is quite difficult to choose randomly a good one.

In that way, a lot of simulations have been done, trying with different number of devices, and it can be concluded that:

- ✓ When working with a configuration of just **one piconet** (no more than 8 devices), the increase for the static algorithm fluctuates normally between **25-35%**, while for the dynamic between **200-225%**.
- ✓ When the configuration is done with **two piconets** (no more than 15 devices), the increase for the static fluctuates normally between **35 and 50%**, while for the dynamic between **225-250%**.
- ✓ And when working with **three piconets** (no more than 22 devices), for the static, in some cases it can even be reached an increase of **75%**, while for the dynamic a **300%** increase.

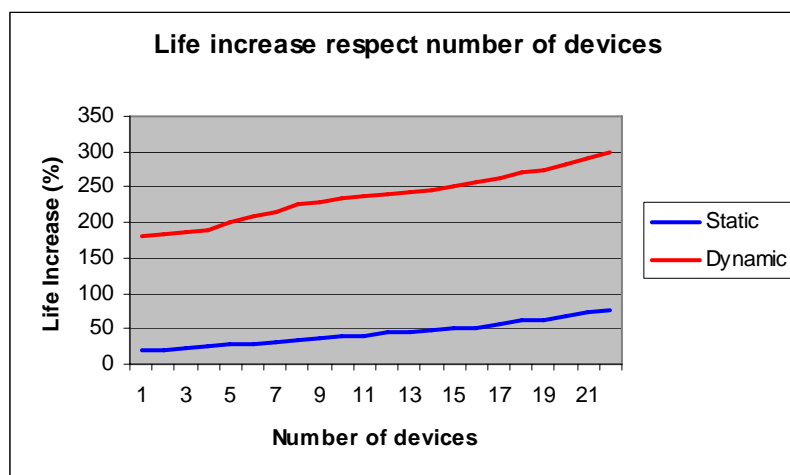


Figure 17: Life increase respect number of devices

- The results of the static algorithm are very good, but the dynamic ones are really spectacular. It is something about 300% increase, which means four times the present life of a configuration. These good results make us think that this is a subject to keep on researching, looking for a definitive creation algorithm that for sure will achieve a battery life optimization in Bluetooth devices.

Example: the results obtained in our example can be seen in the Figure. With the static algorithm it is obtained a 72% increase, while with the dynamic a 280%, which represent really great results.

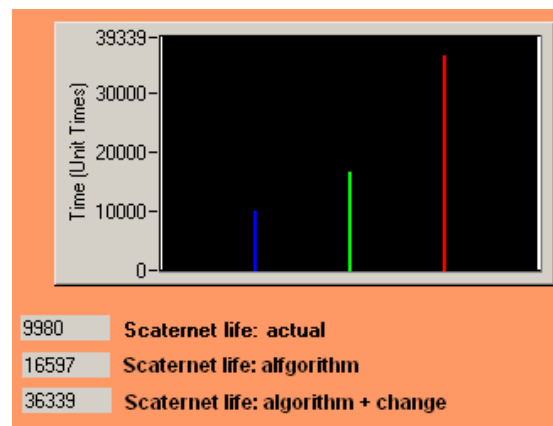


Figure 18: Life configurations

2.4.2. Optimization of the consumption of the Bluetooth devices

It is really interesting to study and analyze the evolution of the devices battery consumption for each configuration: random, static and dynamic. Here it can be found a key to understand why each configuration has so different lives. The evolution for each configuration is analyzed:

▪ **Evolution for the Random Algorithm**

The random configuration works in a static way: once the configuration is created, it will work like that until one device has no battery left. All this means that:

- Considering that both, masters and slaves, have always something to transmit, it can be seen that **the consumption of all the devices is constant along time**, which is something completely logical, because every device is always communicating with the same master (in the case of the slaves), or with the same slaves (in the case of the masters).
 - Analyzing the battery left in the devices at the end of the configuration:
 - ✓ **The configuration always dies because a master** (it could also be a bridge-slave, but it is not too common) has no battery left.
 - ✓ **The masters have spent most of their battery** (or have spent all the battery), **while most of the slaves still have most of their battery left**. The reason for this is that the masters have a higher consumption than the slaves (considering that both, masters and slaves, transmit the same, and they have always something to transmit), because they have to communicate with all their slaves, while one slave only has to communicate with its master.
 - ✓ If it is considered that at the beginning, all the devices have their battery full (%100), calculating at the end, the medium battery left in all the devices, it is obtained a value that fluctuates between 65-75 % of the beginning value: **the piconet/scatternet dies because of battery problems, when the battery left in the whole piconet/scatternet is 65-75 % of the initial value, which indicates a really bad use of the whole piconet/scatternet battery**.
 - In the figure, it can be seen how the medium battery left in the devices is quite high, while the life of the configuration is quite low. With this, it can be concluded that **there is a big amount of battery wasted, and so the low life of the configuration**. In the figure, it can be seen a chart that summarizes the consumption of the devices in this configuration: the devices have 100% of their
-

battery at the beginning, and they have a constant consumption, till one device has no battery left. The medium battery left in the devices fluctuates between 65-75 % of the medium value.

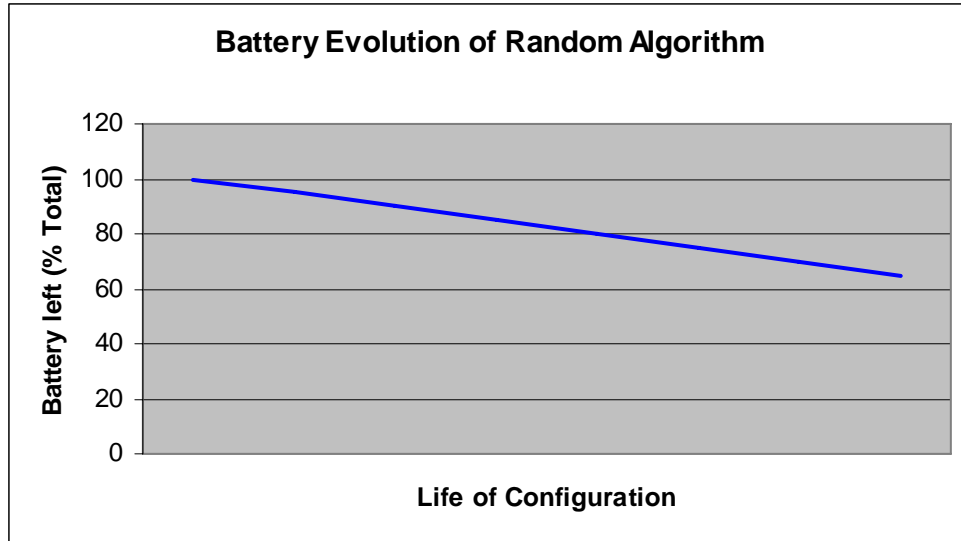


Figure 19: Battery Evolution of Random Algorithm

▪ **Evolution for the Static Algorithm**

The Static Algorithm works in the same way as the Random Configuration, in the way that **the configuration is also static**: once the configuration is created, it will work like that till one device is with no battery. This static behavior makes that this configuration has some similarities from the current configuration:

- The consumption of all the devices is **constant along time**.
- The configuration always **dies because a master** has no battery left.
- At the end of the configuration, the **masters have spent** all or almost spent all their battery, while most of the **slaves still have most of their battery left**.
- The **medium battery left** in the devices, at the end of the configuration, is **65-75 %** of the beginning value, which is more or less the same value obtained for the random configuration.

However, there is a big difference: the life of this configuration is **25-75 % higher** than the random one, **spending more or less the same battery**. That means that just choosing the right configuration, it can be made definitely a better use of the battery, which is really important. All this can be seen in the next chart, where the evolution of the medium battery left (for the random and static configuration) is simulated:

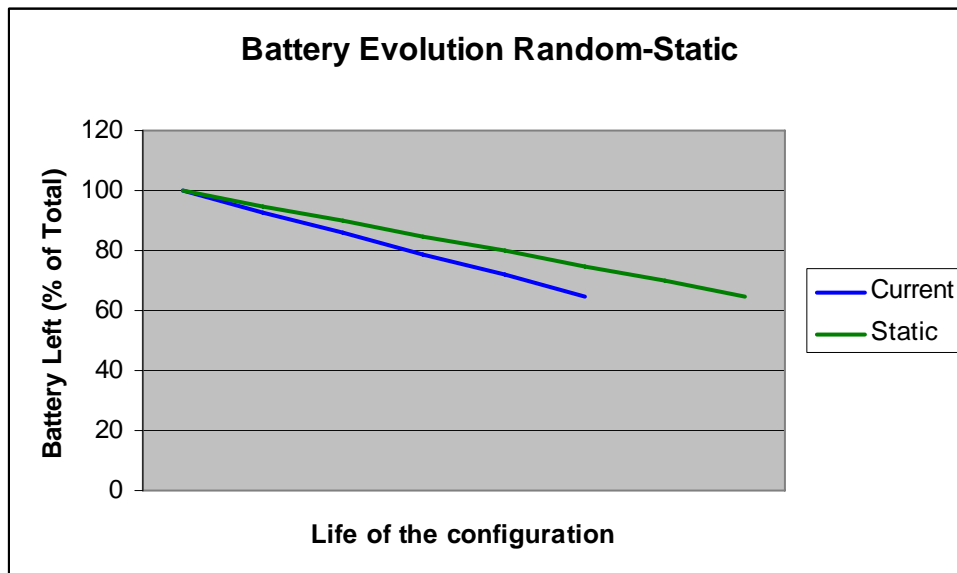


Figure 20: Battery Evolution of Static-Random Algorithm

▪ **Evolution for the Dynamic Algorithm**

Finally, with the dynamic algorithm, it is obtained a non-static configuration but dynamic, where the masters will not always be the same and it will be tried to prolong the life of the configuration spending most of the battery for that goal. It must be emphasized that:

- **The consumption of each device will not be constant along time:** whenever the configuration is changed, every device will be connected to another master, or will begin working as master or whatever. The point is that its consume will be different in every configuration, and so the total will be not constant, although in each configuration it will be constant.
- **The configuration does not always die because of a master:** now the first device spending all its battery may not be a master. Of course it can be one of the masters of the last configuration, but it can also be a slave, that may have worked as a master in a preceding configuration and begins the last configuration with very little battery left.

This point could be thought not to be really important, but it is: when a master dies (spends all its battery) the whole piconet also dies, and maybe the connection between other piconets, if the dead piconet was a bridge between them. If the device dead is a simple slave, the only problem is that that device is not working anymore, but the rest of the piconet and scatternet keeps on working perfectly.

- **At the end of the configuration, most of the devices have spent almost all their battery:** now several configurations will be used and that means that lots of devices will work as masters, and thus they will spend more battery. Therefore, when the configuration dies, it can be easily checked that a lot of devices have spent almost their whole battery, and not just a few ones as it happened in the Random and Static configuration, where the chosen masters worked as masters from the beginning to the end.
- **The medium battery left in all the devices fluctuates between 25% and 30% of the initial value:** this means that it has been spent more or less 40% more of the battery spent in the Random/Static configurations. But the most interesting point of this information, is that increasing the consume in a 40%, it is also obtained a 175-225% increase of the life of the configuration, respect the Static Algorithm, and in a 225-300%, respect the Random Configuration. In the figure 20, it can be seen a comparative chart of the battery evolution of the three configurations.

With these results, it can be concluded that it is really worthy the facts of creating new configurations, spending 40% more of the battery... because in this way, it is obtained an incredible optimization of the battery consume and an increase of the life of the configuration too. All these numbers can be summarized in the next

table, where everything is referred to the results obtained in the simulation of the Current Configuration (Random):

	Static Algorithm	Dynamic Algorithm
Increase of Battery Spent (%)	0%	40%
Increase of the Life of the Configuration (%)	25% - 75%	200% - 300%

Table 2: Comparative Life Increase-Consumption Increase

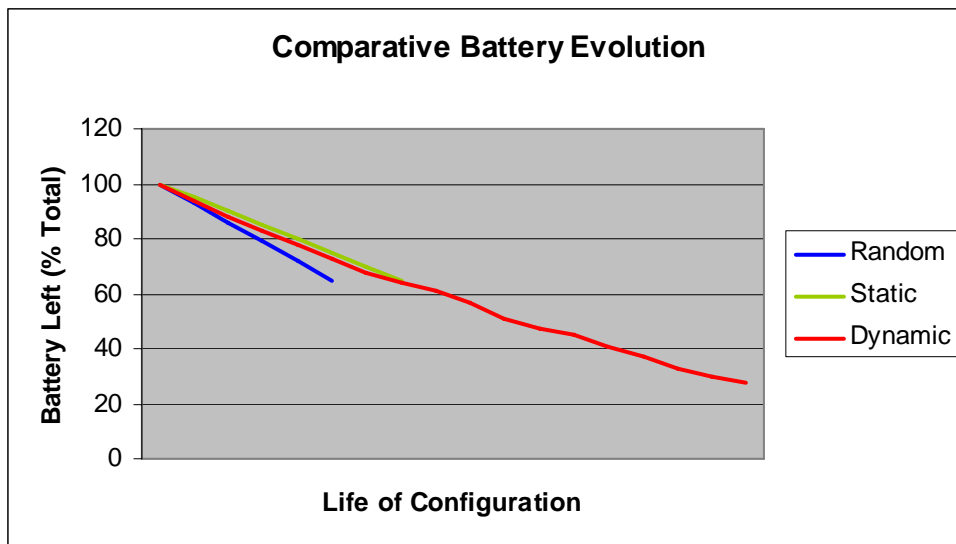


Figure 21: Battery Evolution of the three configurations

Example: in our case, the medium battery left in the devices is: for the random configuration a 69%, for the static a 68% and for the dynamic a 28%, which is a normal result. In the next table, it can be seen these results in comparison with the increase of the life of the configurations:

	Static Algorithm	Dynamic Algorithm
Increase of Battery Spent (%)	1%	39%
Increase of the Life of the Configuration (%)	72%	280%

Table 3: Comparative Life Increase-Consumption Increase example

In order to finish, a chart of a device is shown, where the battery left along time for the three configurations are shown. It can be seen how the random and static configurations have a similar consume, but different lives, while the dynamic has completely different consume and life in comparison with the random and static configuration.

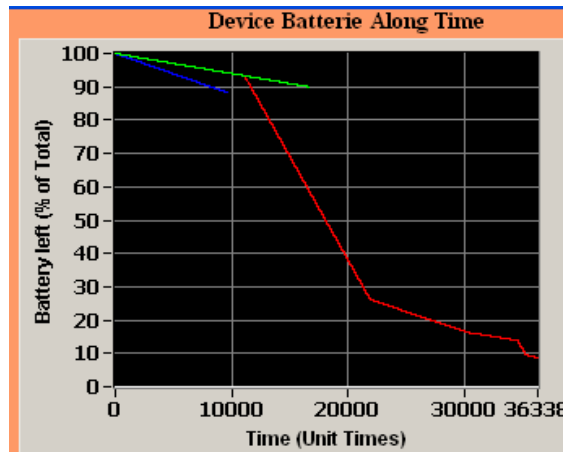


Figure 21: Battery Evolution example

2.4.3. Optimization of the piconet/scatternet configuration

So far, it has been concluded that:

- With the static and dynamic algorithms, it is obtained a huge increase of the life of the configurations.
- A better use of the battery of the devices is done.

Now it is time to find the key for these good results: the optimization of the piconet/scatternet configuration. Actually, the piconets and scatternets are created without following any rule or algorithm, just following a random way. That's why, if a normal configuration is analyzed, it will look some kind of illogical, in the sense that the masters of the piconets are not usually the natural devices to work as that, or there are slaves connected to a master when they have another one much closer to them... making a bad use of the battery with it.

The main goal of this project was to design an algorithm to create the piconets and scatternets in a logical way that made a better use of the batteries and also obtained a battery life optimization. After analyzing the results (increase of the life of the configurations and optimization of the consumption of the devices), it can be concluded that the designed algorithm is a really good one.

Example: the easiest way to show how illogical the random configuration is, (and how logical the configurations of the static and dynamic algorithms are) is just watching the representations of each configuration. It must not be forgotten that the static configuration, and the first of the dynamic, are the same one.

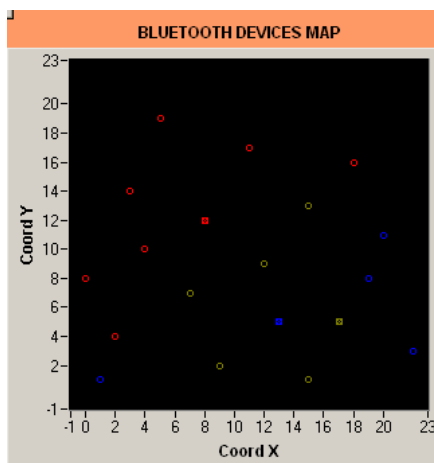


Figure 22: Random Configuration

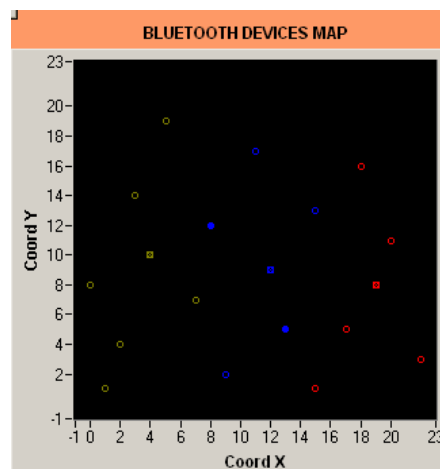


Figure 23: Static Configuration

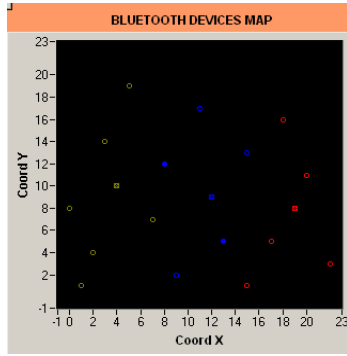


Figure 24: First Dynamic Configuration

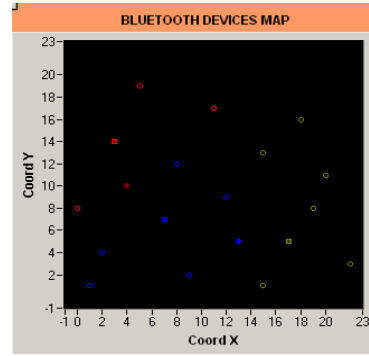


Figure 25: Second Dynamic Configuration

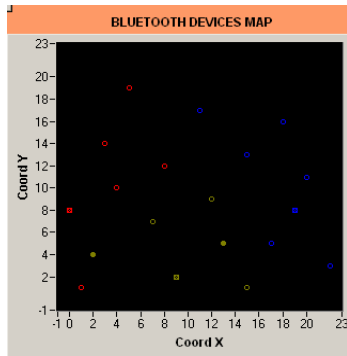


Figure 26: Third Dynamic Configuration

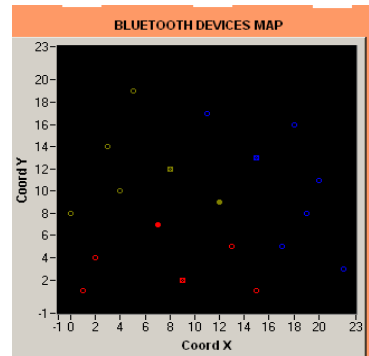


Figure 27: Fourth Dynamic Configuration

2.5. FUTURE STEPS

There is no algorithm defined in the Bluetooth specification for scatternet creation. This means that this model of simulation developed in this project, is just the beginning of a long and interesting work that may be resumed in the future, until this (or another algorithm) is used in the Bluetooth specification for scatternet creation. For future work, it must be considered these two points:

- Other factors to consider
- Hardware Experimenting

2.5.1 Other factors to consider

In this project, it has been developed a quite complete model of simulation, trying to bear in mind as many as possible factors to simulate real Bluetooth behavior. However, because of the lack of time, there are some many other factors that have not been considered and should be also taken into account, because the results obtained considering them will be different for sure, although it must be said, that these changes in the results will not be very significant, because the most important factors have already been considered. These factors to study and analyze in the future are:

- **Consumption during piconet/scatternet formation:** when the piconet/scatternet is being created, the devices have an important consumption while they are in the *Inquiry*, *Inquiry Scan*, *Page* or *Page Scan* states. This consumption is not important for the Static Algorithm because the configuration is created just one time (as it happens in the Current Configuration), but it is really important for the Dynamic Algorithm, because several configurations are created and so the consumption during the creation is multiplied by the number of different configurations used.
 - **Consumption in the different states:** in our model of simulation, it has been considered that the devices were always in the *Active State*, and never in other state. To simulate real Bluetooth behavior, it must also be considered how the consumption changes depending on the Bluetooth state: *standby*, *active*, *hold*, *sniff* and *park* (the inquiry, inquiry scan, page and page scan states have already been considered in the previous point).
 - **Traffic of information of each device:** as said before, in our model, a global situation is considered, where all the devices are always active and have something to transmit. However, the real Bluetooth behavior is not like that: the masters have usually nothing to transmit and the slaves are not always transmitting, and each device has a different amount of information to transmit. So, this global algorithm could be used to create specific ones, depending on the
-

different applications, where the necessity of transmission is different for each one.

- **Piconet operation:** another important factor to consider is the difference between the consumption when the piconet works with an ACL connection and a SCO one.

2.5.2 Hardware Experimenting

Although the results obtained in this project are really good, it must not be forgotten that it is just a model of simulation. The way to prove that this algorithm really works is to experiment with real Bluetooth devices, making Hardware Experiment. The way to make this Hardware Experimenting is the following:

- Every device will be connected to a PC or micro-controller, from where they will be controlled, using the HCI commands. All the PC's and micro-controllers will be connected to a central computer where the algorithm will be executed.
- Using HCI commands, every device will be ordered to look for devices and read the power received from them (RSSI). The device will give this information back to its respective PC or micro-controller.
- The PC, or micro of each device, will transmit the information about the devices seen and RSSI to the central computer. With this information, the algorithm will be executed and the piconet/scatternet configuration will be designed.
- The central computer will send the configuration to every PC or micro, and this will tell its device, using HCI commands. This way the piconet/scatternet will be created.
- Finally, a complete analysis will be done, and these practical results will be compared to the theoretical ones, obtained by simulations.

In the figure, it can be seen a diagram of how to do the Hardware Experimenting:

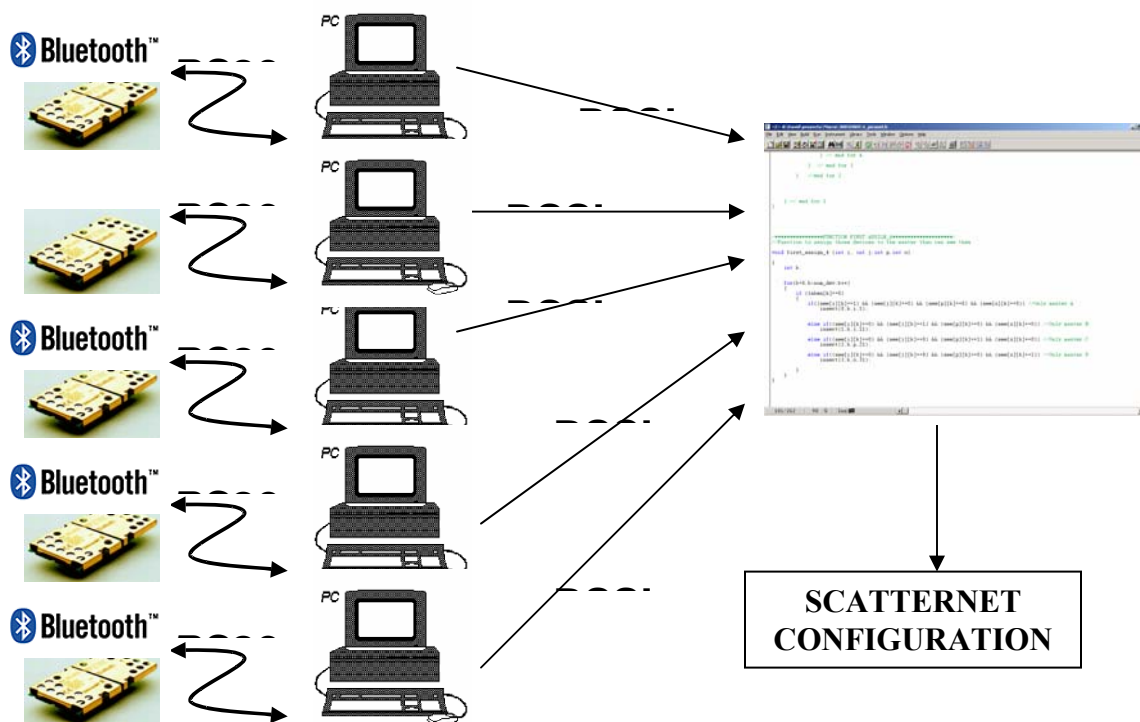


Figure 28: Hardware Experimenting Model

Although, there was not time enough to do the Hardware Experimenting, we began with it, creating a little program in Lab Windows. With this program, from a computer it is possible to control two Bluetooth devices, using the Parallel Ports (one by each port). Using the HCI commands this program can do the next:

- **Reset** the devices (HCI_RESET command).
- A device can be ordered to **discover devices** (HCI_INQUIRY command) and read the BD_ADDRESS of the discovered devices.
- A device can be ordered to **create a connection** with the discovered devices (HCI_CREATE_CONNECTION command), and then **to read the power received** from each one (HCI_READ_RSSI command).
- Finally, the devices can be told to **finish their connections** (using the HCI_DISCONNECT command).

To make easier the use of this program, it has been developed a graphical user interface, where all the tasks explained before can be done. Following, a little manual user is shown, to explain the way in what the program works:

- With the program, **2 devices** can be controlled, one working as master and the other one as slave.
 - The only thing, that can be ordered to the device working as slave is a *reset*. With this reset, the device will return its *bd_address* and it will be printed in the respective number box.
 - The only function of the slave is to have a device controlled, that will be connected to the piconet whose master is the other device controlled. Thus, when both devices are connected, it will be possible to check the results from both sides.
 - With the device working as master, it is possible to:
 - **Make a reset:** the device will return its *bd_address* and it will be printed in the respective number box.
 - **Make an inquiry:** with this inquiry the device will look for discoverable devices. The *bd_addresses* of the discovered devices will be printed in the respective number boxes.
 - **Make a connection:** it can be created a connection with each device discovered (maximum of 7 devices). To do that, it is only needed to press the *Connect* button of the selected device. Once the connection is created a light will be switched on, indicating that the connection is working.
-

- **Read the RSSI:** it can be read the RSSI that the master receives from the devices that have a connection established with it.
- **Make a *disconnection*:** pushing the *disconnect* button, it will be stopped the connection established between the master and the selected device. When the connection is stopped the connection light will be switched off.

In the figure, it can be seen the graphical user interface of this program:

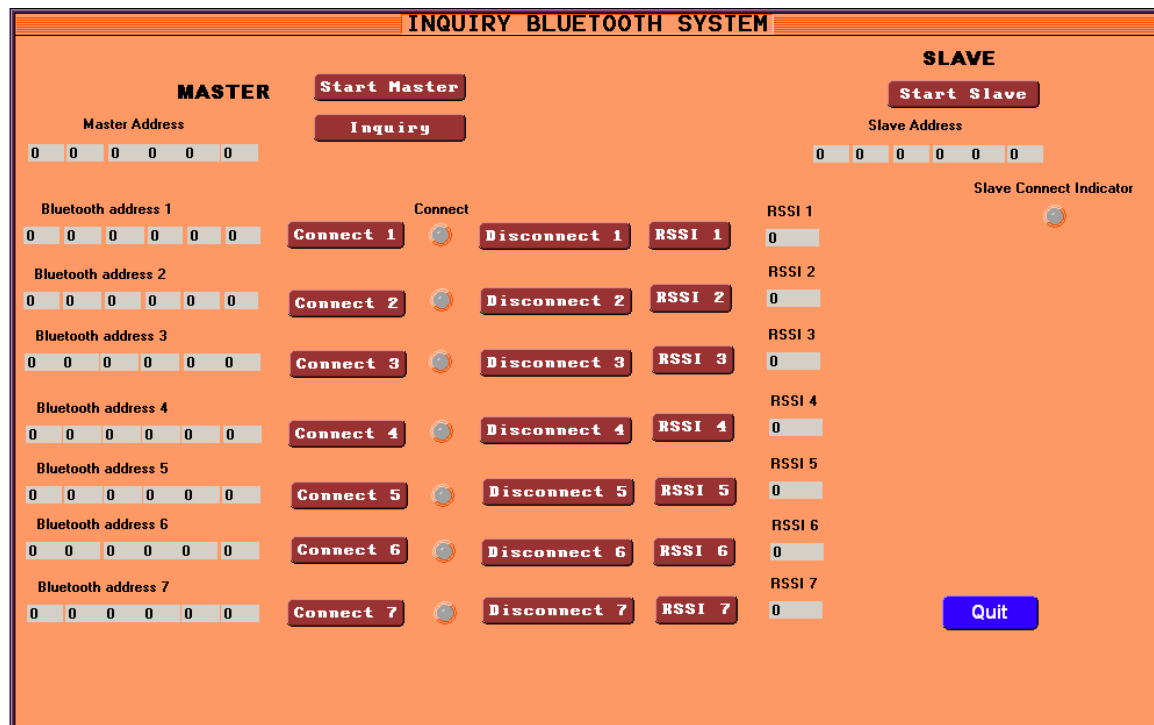


Figure 29: Hardware Experimenting Graphical User Interface

2.6 CONCLUSIONS

After having developed this creation algorithm, analyzed the results of the simulation and explained the steps to be done in the future, it can be concluded the following three points:

2.6.1 Really good results

When this project was begun, we had no idea about the results that could be obtained. Now that it is done, it must be said that they are much better of what could have been thought: an increase in the life of the configuration of 300% (maximum), which is four times the current time.

There are two keys to get these great results: in one hand it is the algorithm developed that works really good, like it has been shown; on the another hand, the point that there is no algorithm defined in the Bluetooth specification. This means that this is a experimental field where nothing has been done yet and it is necessary many things improve.

2.6.2 Need to verify the results of the simulation

Although the results are really good, it must not be forgotten that they are just the offspring of a simulation. To know if the idea of this project is so good, a lot of things have to be done yet: keep on working, making hardware experimenting and bearing in mind all the possible factors that may affect the consumption of the devices.

2.6.3 Keep on working in the future

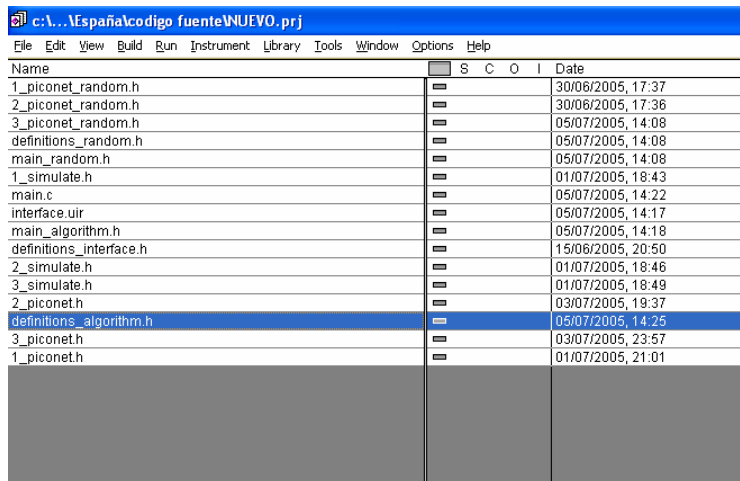
Finally, it must be emphasized that it is really necessary to keep on working on this subject in the future for several reasons:

- ✓ The results of this simulation have been great
 - ✓ There is a need to verify the results of the simulation in normal behavior
 - ✓ There is no algorithm defined in the Bluetooth specification, and so there is a lot to do and to improve
 - ✓ The battery life of the devices is a critical aspect in mobile radio communications. It must be done the best to make the best use of the battery, thus the users have the best final product available.
-

PLANES

3. PLANES

In the following pages, it is shown the commented code files developed for the Algorithm. There are different code files for the creation and simulation of the scatternets made of 1, 2 or 3 piconets. These files have more or less the same code, and so it is not necessary to copy all of them. It has just been included the files for the creation and simulation of one configuration.



The screenshot shows a file explorer window with the following table of files:

Name	S	C	O	I	Date
1_piconet_random.h					30/06/2005, 17:37
2_piconet_random.h					30/06/2005, 17:36
3_piconet_random.h					05/07/2005, 14:08
definitions_random.h					05/07/2005, 14:08
main_random.h					05/07/2005, 14:08
1_simulate.h					01/07/2005, 18:43
main.c					05/07/2005, 14:22
interface.uir					05/07/2005, 14:17
main_algorithm.h					05/07/2005, 14:18
definitions_interface.h					15/06/2005, 20:50
2_simulate.h					01/07/2005, 18:46
3_simulate.h					01/07/2005, 18:49
2_piconeth					03/07/2005, 19:37
definitions_algorithm.h					05/07/2005, 14:25
3_piconeth					03/07/2005, 23:57
1_piconeth					01/07/2005, 21:01

3.1 main.c code

```
//*****MAIN.C*****  
  
//This file is the main one. When we execute the program,  
//first of all we load the graphical interface, from  
//where we control everything and call the rest of files  
//and functions. All the functions in this file are just  
//callback functions that are executed when we press  
//one of the buttons of the graphical interface  
  
#include <cvirte.h>  
#include <userint.h>  
#include <ansi_c.h>  
#include <rs232.h>  
#include <cvirte.h>  
#include <userint.h>  
  
//This is the main file, so we include the rest of the files  
#include "definitions_algorithm.h"  
#include "1_piconet.h"  
#include "2_piconet.h"  
#include "3_piconet.h"  
#include "1_simulate.h"  
#include "2_simulate.h"  
#include "3_simulate.h"  
#include "main_algorithm.h"  
#include "definitions_random.h"  
#include "1_piconet_random.h"  
#include "2_piconet_random.h"  
#include "3_piconet_random.h"  
#include "main_random.h"  
#include "definitions_interface.h"  
  
static int panelHandle;  
  
int main (int argc, char *argv[])  
{  
    if (InitCVIRTE (0, argv, 0) == 0)  
        return -1; /* out of memory */  
    if ((panelHandle = LoadPanel (0, "interface.uir", PANEL)) < 0)  
        return -1;  
    DisplayPanel (panelHandle);  
    RunUserInterface ();  
    DiscardPanel (panelHandle);  
    return 0;  
}
```

```

//****FUNCTION READ NUMBER DEVICES*****
//We use this function to write the number of Bluetooth
//devices we will work with. We write the number of
//the devices in the indicated "number box" in the
//interface and then we press the Enter button. We
//will keeps the value in "num_dev"

int CVICALLBACK read_number_devices (int panel, int control, int event,
    void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

            //We read the value and keeps it in num_dev
            GetCtrlVal (1, PANEL_NUMERIC_2_NUMBER_DEV, &num_dev);

            //We initialize some variable that we
            //will use later: in read_device we
            //just keeps the number of the device
            //that is next to be introduced its
            //coordinates. The first one will be
            //the number one
            read_device=1;
            dev_conf=0;
            SetCtrlVal (1, PANEL_NUMERIC_3_DEV_IND, read_device);

            //We print the caption (leyenda) in
            //the interface, to explain what is
            //a master, what a slave and what a
            //slave-bridge. First we prepare some
            //arrays, and later we print them
            legend_x_master[0]= 2;
            legend_x_slave[0]= 2;
            legend_x_slave_bridge[0]= 2;

            legend_y_master[0]= 1;
            legend_y_slave[0]= 4;
            legend_y_slave_bridge[0]= 7;

            array_legend_x[0]=0;
            array_legend_x[1]=4;
            array_legend_y[0]=0;
            array_legend_y[1]=8;

            PlotXY (1, PANEL_GRAPH_3, legend_x_master,
                legend_y_master,1, VAL_INTEGER, VAL_INTEGER, VAL_SCATTER,
                VAL_EMPTY_SQUARE_WITH_X,VAL_SOLID, 1, VAL_RED);
            PlotXY (1, PANEL_GRAPH_3, legend_x_slave,
                legend_y_slave, 1, VAL_INTEGER, VAL_INTEGER, VAL_SCATTER,
                VAL_EMPTY_CIRCLE,VAL_SOLID, 1, VAL_RED);
            PlotXY (1, PANEL_GRAPH_3,
                legend_x_slave_bridge,legend_y_slave_bridge, 1, VAL_INTEGER,

```

```
VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
VAL_RED);
        PlotXY (1, PANEL_GRAPH_3, array_legend_x,
array_legend_y, 2, VAL_INTEGER, VAL_INTEGER, VAL_SCATTER,
VAL_SIMPLE_DOT,VAL_SOLID, 1, VAL_WHITE);

        break;
    }
    return 0;
}

/**FUNCTION READ COORDINATES*****
//We use this funtcion to write the coordinates of each
//Bluetooth device in space. For each device we will
//write the X and Y coordinate and then press enter. When
//we have written the coordinates of all the devices, our
//algorithm will be run: first we run our algorithm, and
//then a random algorithm to simulate the actual
//behavior of Bluetooth scatternet formation. Then we
//compare both of them, with some graphs and numbers

int CVICALLBACK read_coordinates (int panel, int control, int event,
void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

            //We introduce the values of the coordinates
            //of each device

            if(read_device<=num_dev)
            {
                //We print in a numeric box the number of
                //the device to write its coordinates
                if(read_device!=num_dev)
                    SetCtrlVal (1, PANEL_NUMERIC_3_DEV_IND,
read_device+1);

                //We read the coordinates of the device and
                //keep it coord_x and coord_y
                GetCtrlVal (1, PANEL_NUMERIC_4_COORD_X,
&coord_x[read_device-1]);
                GetCtrlVal (1, PANEL_NUMERIC_5_COORD_Y,
&coord_y[read_device-1]);

                //We reset the numeric boxes where we write
                //the coordinates
                SetCtrlVal (1, PANEL_NUMERIC_4_COORD_X, 0);
                SetCtrlVal (1, PANEL_NUMERIC_5_COORD_Y, 0);
            }
        }
    }
}
```

```
//We prints the device in a map (space):
//first we delete the graph,then we print
//each device in space, and finally
//array_map_x and y so we get a "nice"
//graph. We use the function print_arrays
//to configurate the values of these last
//arrays
print_arrays();
DeleteGraphPlot (1, PANEL_MAP_REPRESENTATION, -
1, VAL_IMMEDIATE_DRAW);
    PlotXY (1, PANEL_MAP_REPRESENTATION, coord_x,
coord_y, read_device, VAL_INTEGER, VAL_INTEGER,VAL_SCATTER,
VAL_DOTTED_SOLID_SQUARE, VAL_SOLID, 1, VAL_WHITE);
    PlotXY (1, PANEL_MAP_REPRESENTATION,
array_map_x, array_map_y, 2,VAL_INTEGER, VAL_INTEGER,
VAL_SCATTER, VAL_SIMPLE_DOT,VAL_SOLID, 1, VAL_BLACK);

//We have introduced the coordinates of one device, so we
//increase the number of devices read (read_device)
    read_device++;

//When we have written the coordinates of all the devices we run
//our algorithm. First we run our algorithm (main_2), and then
//the random algorithm. This will be run so many time as
//so we increase the life time of the scatternet at least in a
    if(read_device>num_dev)
    {

//We run our algorithm and prints the results in the graphs
        main_2();

        do
        {
            main_random();
        }
        while(life_random>(0.85*life_no_change));

//We print the evolution of the batteries of each device with
//time.We will print three graphs: the normal behavior, with
//algorithm and with algorithm and change of configuration
        DeleteGraphPlot (1, PANEL_GRAPH, -1,
VAL_IMMEDIATE_DRAW);
        PlotY (1, PANEL_GRAPH, array_graph[0],
life, VAL_DOUBLE, VAL_THIN_LINE,VAL_EMPTY_SQUARE, VAL_SOLID, 1,
VAL_RED);
        PlotY (1, PANEL_GRAPH,
array_graph_random[0], life_random, VAL_DOUBLE,
VAL_THIN_LINE,VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_BLUE);
        PlotY (1, PANEL_GRAPH,
array_graph_no_change[0], life_no_change, VAL_DOUBLE,
VAL_THIN_LINE,VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_GREEN);
        PlotY (1, PANEL_GRAPH, array_graph_0,
life, VAL_DOUBLE, VAL_THIN_LINE,VAL_EMPTY_SQUARE, VAL_SOLID, 1,
VAL_WHITE);
```

```
        SetCtrlVal (panelHandle, PANEL_NUMERIC
,1);

//We print the Life of the three possibilities: normal behavior,
//algorithm, algorithm and change
        SetCtrlVal (1, PANEL_LIFE_RANDOM,
life_random);
        SetCtrlVal (1, PANEL_LIFE_ALG,
life_no_change);
        SetCtrlVal (1, PANEL_LIFE_ALG_CHANGE,
life);

        //We print in a graph the three lives'
        PlotXY (1,PANEL_GRAPH_2 ,
array_2_normal_x, array_2_normal_y, 2, VAL_INTEGER, VAL_INTEGER,
VAL_VERTICAL_BAR, VAL_DOTTED_EMPTY_DIAMOND, VAL_SOLID, 1,
VAL_BLUE);
        PlotXY (1,PANEL_GRAPH_2 , array_2_alg_x,
array_2_alg_y, 2, VAL_INTEGER, VAL_INTEGER, VAL_VERTICAL_BAR,
VAL_DOTTED_EMPTY_DIAMOND, VAL_SOLID, 1, VAL_GREEN);
        PlotXY (1,PANEL_GRAPH_2 ,
array_2_alg_change_x, array_2_alg_change_y, 2, VAL_INTEGER,
VAL_INTEGER, VAL_VERTICAL_BAR, VAL_DOTTED_EMPTY_DIAMOND,
VAL_SOLID, 1, VAL_RED);
        PlotXY (1,PANEL_GRAPH_2 , array_2_0_x,
array_2_0_y, 2, VAL_INTEGER, VAL_INTEGER,
VAL_BASE_ZERO_VERTICAL_BAR, VAL_DOTTED_EMPTY_DIAMOND, VAL_SOLID,
1, VAL_WHITE);
        PlotXY (1,PANEL_GRAPH_2 , array_2_1_x,
array_2_1_y, 2, VAL_INTEGER, VAL_INTEGER,
VAL_BASE_ZERO_VERTICAL_BAR, VAL_DOTTED_EMPTY_DIAMOND, VAL_SOLID,
1, VAL_WHITE);

        //We print the coordinates of every device
        SetCtrlVal (1, PANEL_COORD_X_1,
coord_x[0]);
        SetCtrlVal (1, PANEL_COORD_X_2,
coord_x[1]);
        SetCtrlVal (1, PANEL_COORD_X_3,
coord_x[2]);
        SetCtrlVal (1, PANEL_COORD_X_4,
coord_x[3]);
        SetCtrlVal (1, PANEL_COORD_X_5,
coord_x[4]);
        SetCtrlVal (1, PANEL_COORD_X_6,
coord_x[5]);
        SetCtrlVal (1, PANEL_COORD_X_7,
coord_x[6]);
        SetCtrlVal (1, PANEL_COORD_X_8,
coord_x[7]);
        SetCtrlVal (1, PANEL_COORD_X_9,
coord_x[8]);
```

```
coord_x[9]);          SetCtrlVal (1, PANEL_COORD_X_10,
coord_x[10]);         SetCtrlVal (1, PANEL_COORD_X_11,
coord_x[11]);         SetCtrlVal (1, PANEL_COORD_X_12,
coord_x[12]);         SetCtrlVal (1, PANEL_COORD_X_13,
coord_x[13]);         SetCtrlVal (1, PANEL_COORD_X_14,
coord_x[14]);         SetCtrlVal (1, PANEL_COORD_X_15,
coord_x[15]);         SetCtrlVal (1, PANEL_COORD_X_16,
coord_x[16]);         SetCtrlVal (1, PANEL_COORD_X_17,
coord_x[17]);         SetCtrlVal (1, PANEL_COORD_X_18,
coord_x[18]);         SetCtrlVal (1, PANEL_COORD_X_19,
coord_x[19]);         SetCtrlVal (1, PANEL_COORD_X_20,
coord_x[20]);         SetCtrlVal (1, PANEL_COORD_X_21,
coord_x[21]);         SetCtrlVal (1, PANEL_COORD_X_22,
coord_y[0]);          SetCtrlVal (1, PANEL_COORD_Y_1,
coord_y[1]);          SetCtrlVal (1, PANEL_COORD_Y_2,
coord_y[2]);          SetCtrlVal (1, PANEL_COORD_Y_3,
coord_y[3]);          SetCtrlVal (1, PANEL_COORD_Y_4,
coord_y[4]);          SetCtrlVal (1, PANEL_COORD_Y_5,
coord_y[5]);          SetCtrlVal (1, PANEL_COORD_Y_6,
coord_y[6]);          SetCtrlVal (1, PANEL_COORD_Y_7,
coord_y[7]);          SetCtrlVal (1, PANEL_COORD_Y_8,
coord_y[8]);          SetCtrlVal (1, PANEL_COORD_Y_9,
coord_y[9]);          SetCtrlVal (1, PANEL_COORD_Y_10,
coord_y[10]);         SetCtrlVal (1, PANEL_COORD_Y_11,
coord_y[11]);         SetCtrlVal (1, PANEL_COORD_Y_12,
coord_y[12]);         SetCtrlVal (1, PANEL_COORD_Y_13,
coord_y[13]);         SetCtrlVal (1, PANEL_COORD_Y_14,
coord_y[14]);         SetCtrlVal (1, PANEL_COORD_Y_15,
```

```
        SetCtrlVal (1, PANEL_COORD_Y_16,
coord_y[15]);
        SetCtrlVal (1, PANEL_COORD_Y_17,
coord_y[16]);
        SetCtrlVal (1, PANEL_COORD_Y_18,
coord_y[17]);
        SetCtrlVal (1, PANEL_COORD_Y_19,
coord_y[18]);
        SetCtrlVal (1, PANEL_COORD_Y_20,
coord_y[19]);
        SetCtrlVal (1, PANEL_COORD_Y_21,
coord_y[20]);
        SetCtrlVal (1, PANEL_COORD_Y_22,
coord_y[21]);

    }

    }
    break;
}
return 0;
}

/*****FUNCTION NEXT*****/
//When we press the NEXT button, we will print in the graph the battery
//evolution in time of the next device, that means that if actually
//we have in the graph the battery evolution of the device 3, after
//pressing the one of device 4. In this graph we will see 3 battery
//charts: one of the random configuration, one of the algorithm and
//the algorithm change one.

int CVICALLBACK next (int panel, int control, int event,
    void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

            if(dev_plot<(num_dev-1))
            {
                dev_plot++;

                //We print the charts
                DeleteGraphPlot (1, PANEL_GRAPH, -1,
VAL_IMMEDIATE_DRAW);
                PlotY (1, PANEL_GRAPH, array_graph[dev_plot], life,
VAL_DOUBLE, VAL_THIN_LINE,VAL_EMPTY_SQUARE, VAL_SOLID, 1,
VAL_RED);
                PlotY (1, PANEL_GRAPH, array_graph_random[dev_plot],
life_random, VAL_DOUBLE, VAL_THIN_LINE,VAL_EMPTY_SQUARE,
VAL_SOLID, 1, VAL_BLUE);
            }
        }
    }
}
```

```
        PlotY (1, PANEL_GRAPH,
array_graph_no_change[dev_plot], life_no_change, VAL_DOUBLE,
VAL_THIN_LINE,VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_GREEN);
        PlotY (1, PANEL_GRAPH, array_graph_0, life,
VAL_DOUBLE, VAL_THIN_LINE,VAL_EMPTY_SQUARE, VAL_SOLID, 1,
VAL_WHITE);

        //We update the number of the device
        SetCtrlVal (panelHandle, PANEL_NUMERIC ,dev_plot+1);
    }

    break;
}
return 0;
}
```

```
/******FUNCTION BEFORE*****
//When we press the BEFORE button, we will print in the graph thbattery
//evolution in time of the previous device, that means that if actually
//we have in the graph the battery evolution of the device 3, after
//pressing the one of device 2. In this graph we will see 3 battery
//charts: one of the random configuration, one of the algorithm and
//the algorithm change one.
```

```
int CVICALLBACK before (int panel, int control, int event,
void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

            if(dev_plot>0)
            {
                dev_plot--;

                //We print the charts
                DeleteGraphPlot (1, PANEL_GRAPH, -1,
VAL_IMMEDIATE_DRAW);
                PlotY (1, PANEL_GRAPH, array_graph[dev_plot], life,
VAL_DOUBLE, VAL_THIN_LINE,VAL_EMPTY_SQUARE, VAL_SOLID, 1,
VAL_RED);
                PlotY (1, PANEL_GRAPH, array_graph_random[dev_plot],
life_random, VAL_DOUBLE, VAL_THIN_LINE,VAL_EMPTY_SQUARE,
VAL_SOLID, 1, VAL_BLUE);
                PlotY (1, PANEL_GRAPH,
array_graph_no_change[dev_plot], life_no_change, VAL_DOUBLE,
VAL_THIN_LINE,VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_GREEN);
                PlotY (1, PANEL_GRAPH, array_graph_0, life,
VAL_DOUBLE, VAL_THIN_LINE,VAL_EMPTY_SQUARE, VAL_SOLID, 1,
VAL_WHITE);

                //We update the number of the device
                SetCtrlVal (panelHandle, PANEL_NUMERIC ,dev_plot+1);
            }
    }
}
```

```
        break;
    }
    return 0;
}

//*****FUNCTION PRINT RANDOM CONFIGURATION*****
//When we press the ACTUAL button, we will print the map of the bluetoh
//devices in space, when we use the random algorithm, and we will also
//write the configuration of the piconets: who the masters is and whohe
//slaves are. The devices of each piconet will be written in differents
//so we can distinguish them

int CVICALLBACK print_random_configuration (int panel, int control, int
    event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

            //We print in the number_boxes, the random configuration, we
            //will indicate who the master of each piconet is, and the
            //slaves of each piconet

                //FIRST PICONET
                SetCtrlVal (1, PANEL_MASTER_1, def_random_pic_1[0]);
                SetCtrlVal (1, PANEL_PICONET_1_1,
def_random_pic_1[1]);
                SetCtrlVal (1, PANEL_PICONET_1_2,
def_random_pic_1[2]);
                SetCtrlVal (1, PANEL_PICONET_1_3,
def_random_pic_1[3]);
                SetCtrlVal (1, PANEL_PICONET_1_4,
def_random_pic_1[4]);
                SetCtrlVal (1, PANEL_PICONET_1_5,
def_random_pic_1[5]);
                SetCtrlVal (1, PANEL_PICONET_1_6,
def_random_pic_1[6]);
                SetCtrlVal (1, PANEL_PICONET_1_7,
def_random_pic_1[7]);

                //SECOND PICONET
                SetCtrlVal (1, PANEL_MASTER_2, def_random_pic_2[0]);
                SetCtrlVal (1, PANEL_PICONET_2_1,
def_random_pic_2[1]);
                SetCtrlVal (1, PANEL_PICONET_2_2,
def_random_pic_2[2]);
                SetCtrlVal (1, PANEL_PICONET_2_3,
def_random_pic_2[3]);
```

```
        SetCtrlVal (1, PANEL_PICONET_2_4,
def_random_pic_2[4]);
        SetCtrlVal (1, PANEL_PICONET_2_5,
def_random_pic_2[5]);
        SetCtrlVal (1, PANEL_PICONET_2_6,
def_random_pic_2[6]);
        SetCtrlVal (1, PANEL_PICONET_2_7,
def_random_pic_2[7]);

//THIRD PICONET
        SetCtrlVal (1, PANEL_MASTER_3, def_random_pic_3[0]);
        SetCtrlVal (1, PANEL_PICONET_3_1,
def_random_pic_3[1]);
        SetCtrlVal (1, PANEL_PICONET_3_2,
def_random_pic_3[2]);
        SetCtrlVal (1, PANEL_PICONET_3_3,
def_random_pic_3[3]);
        SetCtrlVal (1, PANEL_PICONET_3_4,
def_random_pic_3[4]);
        SetCtrlVal (1, PANEL_PICONET_3_5,
def_random_pic_3[5]);
        SetCtrlVal (1, PANEL_PICONET_3_6,
def_random_pic_3[6]);
        SetCtrlVal (1, PANEL_PICONET_3_7,
def_random_pic_3[7]);

//Now we must print the configuration in the map
DeleteGraphPlot (1, PANEL_MAP_REPRESENTATION, -1,
VAL_IMMEDIATE_DRAW);
PlotXY (1, PANEL_MAP_REPRESENTATION, array_map_x,
array_map_y, 2, VAL_INTEGER, VAL_INTEGER, VAL_SCATTER,
VAL_SIMPLE_DOT, VAL_SOLID, 1, VAL_BLACK);

//We print all the devices
if(num_rand_def_1)
    PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_random_pic_1, coord_y_random_pic_1, num_rand_def_1,
VAL_INTEGER, VAL_INTEGER, VAL_SCATTER, VAL_EMPTY_CIRCLE,
VAL_SOLID, 1, VAL_DK_YELLOW);

if(num_rand_def_2)
    PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_random_pic_2, coord_y_random_pic_2, num_rand_def_2,
VAL_INTEGER, VAL_INTEGER, VAL_SCATTER, VAL_EMPTY_CIRCLE,
VAL_SOLID, 1, VAL_RED);

if(num_rand_def_3)
    PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_random_pic_3, coord_y_random_pic_3, num_rand_def_3,
VAL_INTEGER, VAL_INTEGER, VAL_SCATTER, VAL_EMPTY_CIRCLE,
VAL_SOLID, 1, VAL_BLUE);

if(num_rand_def_4)
```

```
        PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_random_pic_4, coord_y_random_pic_4, num_rand_def_4,
VAL_INTEGER, VAL_INTEGER, VAL_SCATTER, VAL_EMPTY_CIRCLE,
VAL_SOLID, 1, VAL_CYAN);

        //We print the masters
        if(num_rand_def_1)
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_random_pic_1, coord_y_random_pic_1, 1, VAL_INTEGER,
VAL_INTEGER, VAL_SCATTER, VAL_EMPTY_SQUARE_WITH_X, VAL_SOLID, 1,
VAL_DK_YELLOW);

        if(num_rand_def_2)
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_random_pic_2, coord_y_random_pic_2, 1, VAL_INTEGER,
VAL_INTEGER, VAL_SCATTER, VAL_EMPTY_SQUARE_WITH_X, VAL_SOLID, 1,
VAL_RED);

        if(num_rand_def_3)
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_random_pic_3, coord_y_random_pic_3, 1, VAL_INTEGER,
VAL_INTEGER, VAL_SCATTER, VAL_EMPTY_SQUARE_WITH_X, VAL_SOLID, 1,
VAL_BLUE);

        if(num_rand_def_4)
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_random_pic_4, coord_y_random_pic_4, 1, VAL_INTEGER,
VAL_INTEGER, VAL_SCATTER, VAL_EMPTY_SQUARE_WITH_X, VAL_SOLID, 1,
VAL_CYAN);

//We print the slaves_bridges (if there are)
//As the bridge belongs to two piconets at the same time, we
//will print it in the color of that piconet which has the
//lowest number of devices
        if(bridge_rand_1_2)
        {
            if(num_rand_def_1 < num_rand_def_2)
                PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_bridge_rand_1_2, coord_y_bridge_rand_1_2, 1,
VAL_INTEGER, VAL_INTEGER, VAL_SCATTER, VAL_SOLID_CIRCLE,
VAL_SOLID, 1, VAL_DK_YELLOW);
            else
                PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_bridge_rand_1_2, coord_y_bridge_rand_1_2, 1,
VAL_INTEGER, VAL_INTEGER, VAL_SCATTER, VAL_SOLID_CIRCLE,
VAL_SOLID, 1, VAL_RED);
        }

        if(bridge_rand_1_3)
        {
            if(num_rand_def_1 < num_rand_def_3)
                PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_bridge_rand_1_3, coord_y_bridge_rand_1_3, 1,
```

```
VAL_INTEGER, VAL_INTEGER, VAL_SCATTER, VAL_SOLID_CIRCLE,
VAL_SOLID, 1, VAL_DK_YELLOW);
    else
        PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_bridge_rand_1_3, coord_y_bridge_rand_1_3, 1,
VAL_INTEGER, VAL_INTEGER, VAL_SCATTER, VAL_SOLID_CIRCLE,
VAL_SOLID, 1, VAL_BLUE);
    }

    if(bridge_rand_2_3)
    {
        if(num_rand_def_2 < num_rand_def_3)
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_bridge_rand_2_3, coord_y_bridge_rand_2_3, 1,
VAL_INTEGER, VAL_INTEGER, VAL_SCATTER, VAL_SOLID_CIRCLE,
VAL_SOLID, 1, VAL_DK_RED);
        else
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_bridge_rand_2_3, coord_y_bridge_rand_2_3, 1,
VAL_INTEGER, VAL_INTEGER, VAL_SCATTER, VAL_SOLID_CIRCLE,
VAL_SOLID, 1, VAL_BLUE);
    }

    break;
}
return 0;
}

/*****FUNCTION PRINT ALG CONFIGURATION*****/
//When we press the ALGORITHM button, we will print the map of the blue
//devices in space, when we use our algorithm without change, or the
//first configuration of the algorithm change, and we will also
//write the configuration of the piconets: who the masters are and who
//slaves are. The devices of each piconet will be written in different
//so we can distinguish them

int CVICALLBACK print_alg_configuration (int panel, int control, int
    event,
    void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

//We print in the number_boxed, the alg configuration, we will indicate
//who the master of each piconet is, and the slaves of each piconet

            //FIRST PICONET
            SetCtrlVal (1, PANEL_MASTER_1, def_alg_pic_1[0][0]);
            SetCtrlVal (1, PANEL_PICONET_1_1,
def_alg_pic_1[0][1]);
```



```
        SetCtrlVal (1, PANEL_PICONET_1_2,
def_alg_pic_1[0][2]);
        SetCtrlVal (1, PANEL_PICONET_1_3,
def_alg_pic_1[0][3]);
        SetCtrlVal (1, PANEL_PICONET_1_4,
def_alg_pic_1[0][4]);
        SetCtrlVal (1, PANEL_PICONET_1_5,
def_alg_pic_1[0][5]);
        SetCtrlVal (1, PANEL_PICONET_1_6,
def_alg_pic_1[0][6]);
        SetCtrlVal (1, PANEL_PICONET_1_7,
def_alg_pic_1[0][7]);

        //SECOND PICONET
        SetCtrlVal (1, PANEL_MASTER_2, def_alg_pic_2[0][0]);
        SetCtrlVal (1, PANEL_PICONET_2_1,
def_alg_pic_2[0][1]);
        SetCtrlVal (1, PANEL_PICONET_2_2,
def_alg_pic_2[0][2]);
        SetCtrlVal (1, PANEL_PICONET_2_3,
def_alg_pic_2[0][3]);
        SetCtrlVal (1, PANEL_PICONET_2_4,
def_alg_pic_2[0][4]);
        SetCtrlVal (1, PANEL_PICONET_2_5,
def_alg_pic_2[0][5]);
        SetCtrlVal (1, PANEL_PICONET_2_6,
def_alg_pic_2[0][6]);
        SetCtrlVal (1, PANEL_PICONET_2_7,
def_alg_pic_2[0][7]);

        //THIRD PICONET
        SetCtrlVal (1, PANEL_MASTER_3, def_alg_pic_3[0][0]);
        SetCtrlVal (1, PANEL_PICONET_3_1,
def_alg_pic_3[0][1]);
        SetCtrlVal (1, PANEL_PICONET_3_2,
def_alg_pic_3[0][2]);
        SetCtrlVal (1, PANEL_PICONET_3_3,
def_alg_pic_3[0][3]);
        SetCtrlVal (1, PANEL_PICONET_3_4,
def_alg_pic_3[0][4]);
        SetCtrlVal (1, PANEL_PICONET_3_5,
def_alg_pic_3[0][5]);
        SetCtrlVal (1, PANEL_PICONET_3_6,
def_alg_pic_3[0][6]);
        SetCtrlVal (1, PANEL_PICONET_3_7,
def_alg_pic_3[0][7]);

        //Now we must print the configuration in the map
        DeleteGraphPlot (1, PANEL_MAP_REPRESENTATION, -1,
VAL_IMMEDIATE_DRAW);
        PlotXY (1, PANEL_MAP_REPRESENTATION, array_map_x,
array_map_y, 2,VAL_INTEGER, VAL_INTEGER, VAL_SCATTER,
VAL_SIMPLE_DOT,VAL_SOLID, 1, VAL_BLACK);

        //We print all the devices
        if(num_alg_def_1[0])
```

```
        PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_1[0], coord_y_alg_pic_1[0], num_alg_def_1[0],
VAL_INTEGER, VAL_INTEGER, VAL_SCATTER, VAL_EMPTY_CIRCLE,
VAL_SOLID, 1, VAL_DK_YELLOW);

        if(num_alg_def_2[0])
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_2[0], coord_y_alg_pic_2[0], num_alg_def_2[0],
VAL_INTEGER, VAL_INTEGER, VAL_SCATTER, VAL_EMPTY_CIRCLE,
VAL_SOLID, 1, VAL_RED);

        if(num_alg_def_3[0])
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_3[0], coord_y_alg_pic_3[0], num_alg_def_3[0],
VAL_INTEGER, VAL_INTEGER, VAL_SCATTER, VAL_EMPTY_CIRCLE,
VAL_SOLID, 1, VAL_BLUE);

        //We print the masters
        if(num_alg_def_1[0])
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_1[0], coord_y_alg_pic_1[0], 1, VAL_INTEGER,
VAL_INTEGER, VAL_SCATTER, VAL_EMPTY_SQUARE_WITH_X, VAL_SOLID, 1,
VAL_DK_YELLOW);

        if(num_alg_def_2[0])
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_2[0], coord_y_alg_pic_2[0], 1, VAL_INTEGER,
VAL_INTEGER, VAL_SCATTER, VAL_EMPTY_SQUARE_WITH_X, VAL_SOLID, 1,
VAL_RED);

        if(num_alg_def_3[0])
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_3[0], coord_y_alg_pic_3[0], 1, VAL_INTEGER,
VAL_INTEGER, VAL_SCATTER, VAL_EMPTY_SQUARE_WITH_X, VAL_SOLID, 1,
VAL_BLUE);

//We print the slaves_bridges (if there are)
//As the bridge belongs to two piconets at the same time, we
//will print it in the color of that piconet which has the
//lowest number of devices
        if(bridge_1_2[0])
        {
            if(num_alg_def_1[0]<num_alg_def_2[0])
                PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_bridge_1_2[0], coord_y_bridge_1_2[0], 1, VAL_INTEGER,
VAL_INTEGER, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
VAL_DK_YELLOW);
            else
                PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_bridge_1_2[0], coord_y_bridge_1_2[0], 1, VAL_INTEGER,
```

```
VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
VAL_RED);
    }

    if(bridge_1_3[0])
    {
        if(num_alg_def_1[0]<num_alg_def_3[0])
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_bridge_1_3[0], coord_y_bridge_1_3[0], 1, VAL_INTEGER,
VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
VAL_DK_YELLOW);
        else
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_bridge_1_3[0], coord_y_bridge_1_3[0], 1, VAL_INTEGER,
VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
VAL_BLUE);
    }

    if(bridge_2_3[0])
    {
        if(num_alg_def_2[0]<num_alg_def_3[0])
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_bridge_2_3[0], coord_y_bridge_2_3[0], 1, VAL_INTEGER,
VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
VAL_DK_RED);
        else
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_bridge_2_3[0], coord_y_bridge_2_3[0], 1, VAL_INTEGER,
VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
VAL_BLUE);
    }

    dev_conf=0;
    break;
}
return 0;
}

/*****FUNCTION PRINT ALG-CHANGE NEXT CONFIGURATION*****/
//When we press the NEXT button, we will print the map of the Bluetooth
//devices in space of the next configuration of the alg + change, and
//write the configuration of the piconets: who the masters are and who
//slaves are. The devices of each piconet will be written in different
//so we can distinguish them

int CVICALLBACK print_alg_change_next (int panel, int control, int
event,
void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
```

```
case EVENT_COMMIT:

    if(dev_conf<(change-1))
    {
        dev_conf++;
//We print in the number_boxed, the alg configuration, we will indicate
//who the master of each piconet is, and the slaves of each piconet

        //FIRST PICONET
        SetCtrlVal (1, PANEL_MASTER_1,
def_alg_pic_1[dev_conf][0]);
        SetCtrlVal (1, PANEL_PICONET_1_1,
def_alg_pic_1[dev_conf][1]);
        SetCtrlVal (1, PANEL_PICONET_1_2,
def_alg_pic_1[dev_conf][2]);
        SetCtrlVal (1, PANEL_PICONET_1_3,
def_alg_pic_1[dev_conf][3]);
        SetCtrlVal (1, PANEL_PICONET_1_4,
def_alg_pic_1[dev_conf][4]);
        SetCtrlVal (1, PANEL_PICONET_1_5,
def_alg_pic_1[dev_conf][5]);
        SetCtrlVal (1, PANEL_PICONET_1_6,
def_alg_pic_1[dev_conf][6]);
        SetCtrlVal (1, PANEL_PICONET_1_7,
def_alg_pic_1[dev_conf][7]);

        //SECOND PICONET
        SetCtrlVal (1, PANEL_MASTER_2,
def_alg_pic_2[dev_conf][0]);
        SetCtrlVal (1, PANEL_PICONET_2_1,
def_alg_pic_2[dev_conf][1]);
        SetCtrlVal (1, PANEL_PICONET_2_2,
def_alg_pic_2[dev_conf][2]);
        SetCtrlVal (1, PANEL_PICONET_2_3,
def_alg_pic_2[dev_conf][3]);
        SetCtrlVal (1, PANEL_PICONET_2_4,
def_alg_pic_2[dev_conf][4]);
        SetCtrlVal (1, PANEL_PICONET_2_5,
def_alg_pic_2[dev_conf][5]);
        SetCtrlVal (1, PANEL_PICONET_2_6,
def_alg_pic_2[dev_conf][6]);
        SetCtrlVal (1, PANEL_PICONET_2_7,
def_alg_pic_2[dev_conf][7]);

        //THIRD PICONET
        SetCtrlVal (1, PANEL_MASTER_3,
def_alg_pic_3[dev_conf][0]);
        SetCtrlVal (1, PANEL_PICONET_3_1,
def_alg_pic_3[dev_conf][1]);
        SetCtrlVal (1, PANEL_PICONET_3_2,
def_alg_pic_3[dev_conf][2]);
        SetCtrlVal (1, PANEL_PICONET_3_3,
def_alg_pic_3[dev_conf][3]);
        SetCtrlVal (1, PANEL_PICONET_3_4,
def_alg_pic_3[dev_conf][4]);
        SetCtrlVal (1, PANEL_PICONET_3_5,
def_alg_pic_3[dev_conf][5]);
```

```
        SetCtrlVal (1, PANEL_PICONET_3_6,
def_alg_pic_3[dev_conf][6]);
        SetCtrlVal (1, PANEL_PICONET_3_7,
def_alg_pic_3[dev_conf][7]);

//Now we must print the configuration in the
map
        DeleteGraphPlot (1, PANEL_MAP_REPRESENTATION, -
1, VAL_IMMEDIATE_DRAW);
        PlotXY (1, PANEL_MAP_REPRESENTATION,
array_map_x, array_map_y, 2,VAL_INTEGER, VAL_INTEGER,
VAL_SCATTER, VAL_SIMPLE_DOT,VAL_SOLID, 1, VAL_BLACK);

//We print all the devices
if(num_alg_def_1[dev_conf])
        PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_1[dev_conf], coord_y_alg_pic_1[dev_conf],
num_alg_def_1[dev_conf], VAL_INTEGER, VAL_INTEGER,VAL_SCATTER,
VAL_EMPTY_CIRCLE, VAL_SOLID, 1, VAL_DK_YELLOW);

if(num_alg_def_2[dev_conf])
        PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_2[dev_conf], coord_y_alg_pic_2[dev_conf],
num_alg_def_2[dev_conf], VAL_INTEGER, VAL_INTEGER,VAL_SCATTER,
VAL_EMPTY_CIRCLE, VAL_SOLID, 1, VAL_RED);

if(num_alg_def_3[dev_conf])
        PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_3[dev_conf], coord_y_alg_pic_3[dev_conf],
num_alg_def_3[dev_conf], VAL_INTEGER, VAL_INTEGER,VAL_SCATTER,
VAL_EMPTY_CIRCLE, VAL_SOLID, 1, VAL_BLUE);

//We print the masters
if(num_alg_def_1[dev_conf])
        PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_1[dev_conf], coord_y_alg_pic_1[dev_conf], 1,
VAL_INTEGER, VAL_INTEGER,VAL_SCATTER, VAL_EMPTY_SQUARE_WITH_X,
VAL_SOLID, 1, VAL_DK_YELLOW);

if(num_alg_def_2[dev_conf])
        PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_2[dev_conf], coord_y_alg_pic_2[dev_conf], 1,
VAL_INTEGER, VAL_INTEGER,VAL_SCATTER, VAL_EMPTY_SQUARE_WITH_X,
VAL_SOLID, 1, VAL_RED);

if(num_alg_def_3[dev_conf])
        PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_3[dev_conf], coord_y_alg_pic_3[dev_conf], 1,
VAL_INTEGER, VAL_INTEGER,VAL_SCATTER, VAL_EMPTY_SQUARE_WITH_X,
VAL_SOLID, 1, VAL_BLUE);
```

```
//We print the slaves_bridges (if there are)
//As the bridge belongs to two piconets at the same time, we will print
//it in the color of that piconet which has the lowest number of vices
    if(bridge_1_2[dev_conf])
    {

        if(num_alg_def_1[dev_conf]<num_alg_def_2[dev_conf])
            PlotXY (1,
                PANEL_MAP_REPRESENTATION, coord_x_bridge_1_2[dev_conf],
                coord_y_bridge_1_2[dev_conf], 1, VAL_INTEGER,
                VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
                VAL_DK_YELLOW);
        else
            PlotXY (1,
                PANEL_MAP_REPRESENTATION, coord_x_bridge_1_2[dev_conf],
                coord_y_bridge_1_2[dev_conf], 1, VAL_INTEGER,
                VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
                VAL_RED);
    }

    if(bridge_1_3[dev_conf])
    {

        if(num_alg_def_1[dev_conf]<num_alg_def_3[dev_conf])
            PlotXY (1,
                PANEL_MAP_REPRESENTATION, coord_x_bridge_1_3[dev_conf],
                coord_y_bridge_1_3[dev_conf], 1, VAL_INTEGER,
                VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
                VAL_DK_YELLOW);
        else
            PlotXY (1,
                PANEL_MAP_REPRESENTATION, coord_x_bridge_1_3[dev_conf],
                coord_y_bridge_1_3[dev_conf], 1, VAL_INTEGER,
                VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
                VAL_BLUE);
    }

    if(bridge_2_3[dev_conf])
    {

        if(num_alg_def_2[dev_conf]<num_alg_def_3[dev_conf])
            PlotXY (1,
                PANEL_MAP_REPRESENTATION, coord_x_bridge_2_3[dev_conf],
                coord_y_bridge_2_3[dev_conf], 1, VAL_INTEGER,
                VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
                VAL_DK_RED);
        else
            PlotXY (1,
                PANEL_MAP_REPRESENTATION, coord_x_bridge_2_3[dev_conf],
                coord_y_bridge_2_3[dev_conf], 1, VAL_INTEGER,
                VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
                VAL_BLUE);
    }
}
```

```
        }

        break;
    }
    return 0;
}

//*****FUNCTION PRINT ALG-CHANGE BEFORE CONFIGURATION*****
//When we press the BEFORE button, we will print the map of the bluet
//devices in space of the previous configuration of the alg + change,
//write the configuration of the piconets: who the masters are and who
//slaves are. The devices of each piconet will be written in different
//so we can distinguish them

int CVICALLBACK print_alg_change_before (int panel, int control, int
    event,
    void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

            if(dev_conf>0)
            {
                dev_conf--;
                //We print in the number_boxes, the alg configuration, we will indicate
                //who the master of each piconet is, and the slaves of each piconet

                //FIRST PICONET
                SetCtrlVal (1, PANEL_MASTER_1,
def_alg_pic_1[dev_conf][0]);
                SetCtrlVal (1, PANEL_PICONET_1_1,
def_alg_pic_1[dev_conf][1]);
                SetCtrlVal (1, PANEL_PICONET_1_2,
def_alg_pic_1[dev_conf][2]);
                SetCtrlVal (1, PANEL_PICONET_1_3,
def_alg_pic_1[dev_conf][3]);
                SetCtrlVal (1, PANEL_PICONET_1_4,
def_alg_pic_1[dev_conf][4]);
                SetCtrlVal (1, PANEL_PICONET_1_5,
def_alg_pic_1[dev_conf][5]);
                SetCtrlVal (1, PANEL_PICONET_1_6,
def_alg_pic_1[dev_conf][6]);
                SetCtrlVal (1, PANEL_PICONET_1_7,
def_alg_pic_1[dev_conf][7]);

                //SECOND PICONET
                SetCtrlVal (1, PANEL_MASTER_2,
def_alg_pic_2[dev_conf][0]);
                SetCtrlVal (1, PANEL_PICONET_2_1,
def_alg_pic_2[dev_conf][1]);
```

```
        SetCtrlVal (1, PANEL_PICONET_2_2,
def_alg_pic_2[dev_conf][2]);
        SetCtrlVal (1, PANEL_PICONET_2_3,
def_alg_pic_2[dev_conf][3]);
        SetCtrlVal (1, PANEL_PICONET_2_4,
def_alg_pic_2[dev_conf][4]);
        SetCtrlVal (1, PANEL_PICONET_2_5,
def_alg_pic_2[dev_conf][5]);
        SetCtrlVal (1, PANEL_PICONET_2_6,
def_alg_pic_2[dev_conf][6]);
        SetCtrlVal (1, PANEL_PICONET_2_7,
def_alg_pic_2[dev_conf][7]);

        //THIRD PICONET
        SetCtrlVal (1, PANEL_MASTER_3,
def_alg_pic_3[dev_conf][0]);
        SetCtrlVal (1, PANEL_PICONET_3_1,
def_alg_pic_3[dev_conf][1]);
        SetCtrlVal (1, PANEL_PICONET_3_2,
def_alg_pic_3[dev_conf][2]);
        SetCtrlVal (1, PANEL_PICONET_3_3,
def_alg_pic_3[dev_conf][3]);
        SetCtrlVal (1, PANEL_PICONET_3_4,
def_alg_pic_3[dev_conf][4]);
        SetCtrlVal (1, PANEL_PICONET_3_5,
def_alg_pic_3[dev_conf][5]);
        SetCtrlVal (1, PANEL_PICONET_3_6,
def_alg_pic_3[dev_conf][6]);
        SetCtrlVal (1, PANEL_PICONET_3_7,
def_alg_pic_3[dev_conf][7]);

        //Now we must print the configuration in the
map
        DeleteGraphPlot (1, PANEL_MAP_REPRESENTATION, -
1, VAL_IMMEDIATE_DRAW);
        PlotXY (1, PANEL_MAP_REPRESENTATION,
array_map_x, array_map_y, 2,VAL_INTEGER, VAL_INTEGER,
VAL_SCATTER, VAL_SIMPLE_DOT,VAL_SOLID, 1, VAL_BLACK);

        //We print all the devices
        if(num_alg_def_1[dev_conf])
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_1[dev_conf], coord_y_alg_pic_1[dev_conf],
num_alg_def_1[dev_conf], VAL_INTEGER, VAL_INTEGER,VAL_SCATTER,
VAL_EMPTY_CIRCLE, VAL_SOLID, 1, VAL_DK_YELLOW);

            if(num_alg_def_2[dev_conf])
                PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_2[dev_conf], coord_y_alg_pic_2[dev_conf],
num_alg_def_2[dev_conf], VAL_INTEGER, VAL_INTEGER,VAL_SCATTER,
VAL_EMPTY_CIRCLE, VAL_SOLID, 1, VAL_RED);

            if(num_alg_def_3[dev_conf])
```



```
        PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_3[dev_conf], coord_y_alg_pic_3[dev_conf],
num_alg_def_3[dev_conf], VAL_INTEGER, VAL_INTEGER,VAL_SCATTER,
VAL_EMPTY_CIRCLE, VAL_SOLID, 1, VAL_BLUE);

        //We print the masters
        if(num_alg_def_1[dev_conf])
            PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_1[dev_conf], coord_y_alg_pic_1[dev_conf], 1,
VAL_INTEGER, VAL_INTEGER,VAL_SCATTER, VAL_EMPTY_SQUARE_WITH_X,
VAL_SOLID, 1, VAL_DK_YELLOW);

            if(num_alg_def_2[dev_conf])
                PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_2[dev_conf], coord_y_alg_pic_2[dev_conf], 1,
VAL_INTEGER, VAL_INTEGER,VAL_SCATTER, VAL_EMPTY_SQUARE_WITH_X,
VAL_SOLID, 1, VAL_RED);

                    if(num_alg_def_3[dev_conf])
                        PlotXY (1, PANEL_MAP_REPRESENTATION,
coord_x_alg_pic_3[dev_conf], coord_y_alg_pic_3[dev_conf], 1,
VAL_INTEGER, VAL_INTEGER,VAL_SCATTER, VAL_EMPTY_SQUARE_WITH_X,
VAL_SOLID, 1, VAL_BLUE);

//We print the slaves_bridges (if there are)
//As the bridge belongs to two piconets at the same time, we will print
//it in the color of that piconet which has the lowest number of
        if(bridge_1_2[dev_conf])
            {

                if(num_alg_def_1[dev_conf]<num_alg_def_2[dev_conf])
                    PlotXY (1,
PANEL_MAP_REPRESENTATION, coord_x_bridge_1_2[dev_conf],
coord_y_bridge_1_2[dev_conf], 1, VAL_INTEGER,
VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
VAL_DK_YELLOW);
                    else
                        PlotXY (1,
PANEL_MAP_REPRESENTATION, coord_x_bridge_1_2[dev_conf],
coord_y_bridge_1_2[dev_conf], 1, VAL_INTEGER,
VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
VAL_RED);
            }

                if(bridge_1_3[dev_conf])
                    {

                        if(num_alg_def_1[dev_conf]<num_alg_def_3[dev_conf])
                            PlotXY (1,
PANEL_MAP_REPRESENTATION, coord_x_bridge_1_3[dev_conf],
coord_y_bridge_1_3[dev_conf], 1, VAL_INTEGER,
```

```
VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
VAL_DK_YELLOW);
        else
            PlotXY (1,
                PANEL_MAP_REPRESENTATION, coord_x_bridge_1_3[dev_conf],
                coord_y_bridge_1_3[dev_conf], 1, VAL_INTEGER,
                VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
                VAL_BLUE);
        }

        if(bridge_2_3[dev_conf])
        {

            if(num_alg_def_2[dev_conf]<num_alg_def_3[dev_conf])
                PlotXY (1,
                    PANEL_MAP_REPRESENTATION, coord_x_bridge_2_3[dev_conf],
                    coord_y_bridge_2_3[dev_conf], 1, VAL_INTEGER,
                    VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
                    VAL_DK_RED);
            else
                PlotXY (1,
                    PANEL_MAP_REPRESENTATION, coord_x_bridge_2_3[dev_conf],
                    coord_y_bridge_2_3[dev_conf], 1, VAL_INTEGER,
                    VAL_INTEGER,VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID, 1,
                    VAL_BLUE);
        }

    }
    break;
}
return 0;
}
```

```
//*****FUNCTION SALIR*****
//Function to finish the program
```

```
int CVICALLBACK salir (int panel, int control, int event,
    void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            QuitUserInterface (0);
            break;
    }
    return 0;
}
```

3.2 definitions interface.h

```
/* *****  
/* LabWindows/CVI User Interface Resource (UIR) Include File  
/* Copyright (c) National Instruments 2005. All Rights Reserved.  
/* WARNING: Do not add to, delete from, or otherwise modify the      */  
/*           of this include file.  
/* *****  
  
#include <userint.h>  
  
#ifdef __cplusplus  
    extern "C" {  
#endif  
  
    /* Panels and Controls: */  
  
#define PANEL 1  
#define PANEL_COMMANDBUTTON_2 2  
#define PANEL_NUMERIC 3  
#define PANEL_COMMANDBUTTON_4 4  
#define PANEL_COMMANDBUTTON_3 5  
#define PANEL_TEXTBOX_NUMBER_DEVICE 6  
#define PANEL_NUMERIC_2_NUMBER_DEV 7  
#define PANEL_COMMANDBUTTON_6_ENT_2 8  
#define PANEL_COMMANDBUTTON_5_ENT_1 9  
#define PANEL_NUMERIC_3_DEV_IND 10  
#define PANEL_NUMERIC_5_COORD_Y 11  
#define PANEL_NUMERIC_4_COORD_X 12  
#define PANEL_LIFE_RANDOM 13  
#define PANEL_LIFE_ALG 14  
#define PANEL_LIFE_ALG_CHANGE 15  
#define PANEL_MAP_REPRESENTATION 16  
#define PANEL_MAP_CHANGE_2 17  
#define PANEL_MAP_CHANGE_1 18  
#define PANEL_MAP_ALGORITHM 19  
#define PANEL_MAP_RANDOM 20  
#define PANEL_BLUETOOTH_NUMBER_22 21  
#define PANEL_BLUETOOTH_NUMBER_21 22  
#define PANEL_BLUETOOTH_NUMBER_20 23  
#define PANEL_BLUETOOTH_NUMBER_19 24  
#define PANEL_BLUETOOTH_NUMBER_18 25  
#define PANEL_BLUETOOTH_NUMBER_17 26  
#define PANEL_BLUETOOTH_NUMBER_16 27  
#define PANEL_BLUETOOTH_NUMBER_15 28  
#define PANEL_BLUETOOTH_NUMBER_14 29
```

```
#define PANEL_BLUETOOTH_NUMBER_13 30
#define PANEL_BLUETOOTH_NUMBER_12 31
#define PANEL_BLUETOOTH_NUMBER_11 32
#define PANEL_BLUETOOTH_NUMBER_10 33
#define PANEL_BLUETOOTH_NUMBER_9 34
#define PANEL_BLUETOOTH_NUMBER_8 35
#define PANEL_BLUETOOTH_NUMBER_7 36
#define PANEL_BLUETOOTH_NUMBER_6 37
#define PANEL_BLUETOOTH_NUMBER_5 38
#define PANEL_BLUETOOTH_NUMBER_4 39
#define PANEL_BLUETOOTH_NUMBER_3 40
#define PANEL_BLUETOOTH_NUMBER_2 41
#define PANEL_COORD_Y_22 42
#define PANEL_COORD_Y_21 43
#define PANEL_COORD_Y_20 44
#define PANEL_COORD_Y_19 45
#define PANEL_COORD_Y_18 46
#define PANEL_COORD_Y_17 47
#define PANEL_COORD_Y_16 48
#define PANEL_COORD_Y_15 49
#define PANEL_COORD_Y_14 50
#define PANEL_COORD_Y_13 51
#define PANEL_COORD_Y_12 52
#define PANEL_COORD_Y_11 53
#define PANEL_COORD_Y_10 54
#define PANEL_COORD_Y_9 55
#define PANEL_COORD_Y_8 56
#define PANEL_COORD_Y_7 57
#define PANEL_COORD_Y_6 58
#define PANEL_COORD_Y_5 59
#define PANEL_COORD_Y_4 60
#define PANEL_COORD_Y_3 61
#define PANEL_COORD_Y_2 62
#define PANEL_COORD_Y_1 63
#define PANEL_COORD_X_22 64
#define PANEL_COORD_X_21 65
#define PANEL_COORD_X_20 66
#define PANEL_COORD_X_19 67
#define PANEL_COORD_X_18 68
#define PANEL_COORD_X_17 69
#define PANEL_COORD_X_16 70
#define PANEL_COORD_X_15 71
#define PANEL_COORD_X_14 72
#define PANEL_COORD_X_13 73
#define PANEL_COORD_X_12 74
#define PANEL_COORD_X_11 75
#define PANEL_COORD_X_10 76
#define PANEL_COORD_X_9 77
#define PANEL_COORD_X_8 78
#define PANEL_COORD_X_7 79
#define PANEL_COORD_X_6 80
#define PANEL_COORD_X_5 81
#define PANEL_COORD_X_4 82
#define PANEL_COORD_X_3 83
#define PANEL_COORD_X_2 84
#define PANEL_COORD_X_1 85
#define PANEL_BLUETOOTH_NUMBER_1 86
```

```
#define PANEL_PICONET_3_6 87
#define PANEL_PICONET_3_7 88
#define PANEL_PICONET_3_5 89
#define PANEL_PICONET_3_4 90
#define PANEL_PICONET_3_3 91
#define PANEL_PICONET_3_2 92
#define PANEL_PICONET_3_1 93
#define PANEL_PICONET_2_7 94
#define PANEL_PICONET_2_6 95
#define PANEL_PICONET_2_5 96
#define PANEL_PICONET_2_4 97
#define PANEL_PICONET_2_3 98
#define PANEL_PICONET_2_2 99
#define PANEL_PICONET_2_1 100
#define PANEL_PICONET_1_7 101
#define PANEL_PICONET_1_6 102
#define PANEL_PICONET_1_5 103
#define PANEL_PICONET_1_4 104
#define PANEL_PICONET_1_3 105
#define PANEL_PICONET_1_2 106
#define PANEL_PICONET_1_1 107
#define PANEL_MASTER_3 108
#define PANEL_MASTER_2 109
#define PANEL_MASTER_1 110
#define PANEL_GRAPH 111
#define PANEL_GRAPH_2 112
#define PANEL_GRAPH_3 113
#define PANEL_TEXTMSG_2 114
#define PANEL_TEXTMSG 115
#define PANEL_TEXTMSG_3 116

/* Menu Bars, Menus, and Menu Items: */

/* (no menu bars in the resource file) */

/* Callback Prototypes: */

int CVICALLBACK before(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int CVICALLBACK next(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int CVICALLBACK print_alg_change_before(int panel, int control, int
event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK print_alg_change_next(int panel, int control, int
event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK print_alg_configuration(int panel, int control, int
event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK print_random_configuration(int panel, int control, int
event, void *callbackData, int eventData1, int eventData2);
int CVICALLBACK read_coordinates(int panel, int control, int event,
void *callbackData, int eventData1, int eventData2);
int CVICALLBACK read_number_devices(int panel, int control, int event,
void *callbackData, int eventData1, int eventData2);
int CVICALLBACK salir(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
```

```
#ifdef __cplusplus
    }
#endif
```

3.3 main algorithm.h

```
/******MAIN_ALGORITHM.H*****

//This is the file where we will execute algorithm: we will create
//out scatternet, and simulate its behavior in time: we will calculate
//the lifetime and the evolution of the battery of each device along

/******

/******FUNCION MAIN 2*****/
//Main function where we will calculate the distances between
//each couple of devices, the received power by each device from
//the rest, the transmitted power by one device to the others, and
//using all these results we will run our algorithm and get
//the scatternet configuration. Finally we will simulate it and
//get the lifetime and the evolution of the batteries along time

void main_2 ()
{
    int i,j,p,k;

    calc_distances(); //We calculate the distance
    calc_rx_power(); //We calculate the rx power
    calc_tx_power(); //We estimate the tx power
    reset_1(); //We reset and initialize
    run_alg(); //We run our algorithm
    reset_2(); //We reset and inicializa some variables,
              //so they are ready
              //ready to be used by the simulation
              //algorithm function
    simulate_alg(); //We simulate the results
                  //and get the arrays to print

    reset_3();

} //End of main

/******FUNCTION PRINT_ARRAYS*****
//Function to create the arrays that we are gonna print.
```

```
//The graph of the different lives of the scatternets, and
//also to print a map with the devices in space

void print_arrays(void)
{
    int i, min_x, min_y, max_x, max_y;

    //We create some others arrays, to have a nice representation

    min_x=10;
    min_y=10;
    max_x=0;
    max_y=0;

    for(i=0;i<num_dev;i++)
    {
        //We look the min_x
        if(coord_x[i]<min_x)
            min_x=coord_x[i];

        //We look the min_y
        if(coord_y[i]<min_y)
            min_y=coord_y[i];

        //We look the max_x
        if(coord_x[i]>max_x)
            max_x=coord_x[i];

        //We look the max_y
        if(coord_y[i]>max_y)
            max_y=coord_y[i];
    }

    if(min_x<=min_y)
        min=min_x;
    else
        min=min_y;

    if(max_x>=max_y)
        max=max_x;
    else
        max=max_y;

    //We create two arrays with values a bit smaller and bigger
    //the devices, so the representation in the map is nicer
    array_map_x[0]=max+1;
    array_map_x[1]=min-1;
    array_map_y[0]=max+1;
    array_map_y[1]=min-1;

}
```

```

/*****FUNCION CALC_DISTANCES*****/
//Function to calculate the distances between every couple of devices
//using Pitagoras In dist[i][j] we will keep the distance in distance
//units between two devices i and j

void calc_distances ()
{
    int i,j,a,b;
    double c;

    for (i=0;i<num_dev;i++)
    {
        for (j=0;j<num_dev;j++)
        {
            a=abs(coord_x[i]-coord_x[j]);
            b=abs(coord_y[i]-coord_y[j]);
            a=pow(a,2);
            b=pow(b,2);
            c=sqrt(a+b);
            dist[i][j]=c;
        }
    }
}

/*****FUNCION CALC_RX_POWER*****/
//Function to calculate the received power by each device from the rest
//using Friis. In power[i][j] we will keep the power
//received by the device i by the device j in dBm if you connect them

void calc_rx_power ()
{
    int i,j;

    for (i=0;i<num_dev;i++)
    {
        see[i][i]=0;

        for (j=0;j<num_dev;j++)
        {
            if (i!=j)
            {
                power[i][j]=(5+(20*(log(0.5)))-
(20*(log(dist[i][j])))); //power in dBm

                if (power[i][j]>=-60)
                {
                    power[i][j]=(power[i][j])/20; //We change
to natural values
                    power[i][j]=pow(10,power[i][j]);
                }

                see[i][j]=1;

                power[i][num_dev]=power[i][num_dev]+power[i][j];
            }
        }
    }
}

```

```
        }
        else //If we rx less than 60 dbm, we can see
that device
        {
        //printf("\n\tdesde el Bluetooth %d: no
se ve", j);
        see[i][j]=0;
        }
    }
}

//printf("\n\n\n");
//scanf ("%d", &prueba);
}
```

```
/******FUNCION CALC_TX_POWER******/
//Function to estimate the transmitted power by each device in
//0.36 mW is the maximum received power. In tx_power[i][j] we will
//keep the tx_power by device i to device j if you connect them
```

```
void calc_tx_power ()
{
    int i,j;

    for (i=0;i<num_dev;i++)
    {
        for (j=0;j<num_dev;j++)
            tx_power[i][j]=0.360456-power[i][j];
    }
}
```

```
/******FUNCION RESET_1******/
//Function to reset and initialize some variables and arrays
```

```
void reset_1 (void)
{
    int i,j,p,k;

    dead=0,dead_2=0,life=0, life_no_change=0,change=0,change_2=0;

    //We begin the batterie values of the devices
    for(i=0;i<num_dev;i++)
        batterie[i]=100000;
```

```
//We check that the devices still have batteries, so they are
for(i=0;i<num_dev;i++)
    if(batterie[i]<0)
        dead=1;

//We reset the def_alg_pic arrays
for(i=0;i<50;i++)
{
    num_alg_def_1[i]=0;
    num_alg_def_2[i]=0;
    num_alg_def_3[i]=0;

    for(j=0;j<8;j++)
    {
        def_alg_pic_1[i][j]=0;
        def_alg_pic_2[i][j]=0;
        def_alg_pic_3[i][j]=0;
        coord_x_bridge_1_2[i][j]=0;
        coord_x_bridge_2_3[i][j]=0;
        coord_x_bridge_1_3[i][j]=0;
        coord_y_bridge_1_2[i][j]=0;
        coord_y_bridge_2_3[i][j]=0;
        coord_y_bridge_1_3[i][j]=0;
    }
}

//We reset the configurations arrays where we keep the lifetime of the
//scatternet of every different configuration
for(i=0;i<8;i++)
    configurations_1[i]=0;

for(i=0;i<20;i++)
{
    used_conf_1[i]=0;
    used_conf_2_1[i]=0;
    used_conf_2_2[i]=0;
    used_conf_3_1[i]=0;
    used_conf_3_2[i]=0;
    used_conf_3_3[i]=0;
}

for(i=0;i<15;i++)
    for(j=0;j<15;j++)
        configurations_2[i][j]=0;

for(i=0;i<22;i++)
    for(j=0;j<22;j++)
        for(p=0;p<22;p++)
            configurations_3[i][j][p]=0;

//We reset the indexes and arrays that we use to simulate the
//life of scatternet, to get
//later the graphs
```

```
ind_conf_1=0;
ind_conf_2=0;
ind_conf_3=0;

//We reset the array_graph array
for(i=0;i<num_dev;i++)
    for(j=0;j<100000;j++)
        array_graph_no_change[i][j]=0;

for(i=0;i<num_dev;i++)
    array_graph_no_change[i][0]=100;
}

/*****FUNCION RUN ALGORITHM*****/
//Function to run our algorithm: to save batteries, our algorithm will
//the lowest number of piconets. So, if we have till 8 devices, we will
//one piconet, if we have till 15 we will try 2 piconets, and if we
//till 22 we will try with three piconets. Of course, if the configura
//is not possible with the lowest number of piconets, we will try with

void run_alg (void)
{
    int i;

    //We beguin to form the scatternet configuration
    if (num_dev<=8) //If we have less than 8 devices we
        //will try just one piconet
    {
        while (dead==0)
        {
            for(i=0;i<num_dev;i++)
                if(batterie[i]<=0)
                    dead=1;
            if(dead==0)
                piconet_1();
        }
    }

    else if (num_dev<16 && num_dev>8)
//If we have 9-15 devices, we will try two piconets
    {
        do
        {
            for(i=0;i<num_dev;i++)
                if(batterie[i]<=0)
                    dead_2=1;
        }
    }
}
```

```
        if(dead_2==0)
            piconet_2();
    }
    while ((dead==0)&&(dead_2==0));
}

else if (num_dev<23 && num_dev>15)
//We will try three piconets
{
    do
    {
        for(i=0;i<num_dev;i++)
            if(batterie[i]<=0)
                dead_2=1;
        if(dead_2==0)
            piconet_3();
    }
    while ((dead==0)&&(dead_2==0));
}

//We print the total life of the scatternet
//printf("\n\n\nTotal life of scatternet: %d ",life);

//printf("\n\n\n");
//scanf ("%d", &prueba);
}

/*****FUNCION RESET_2*****/
//Function to reset and initialize some variables and arrays, that we
//are gonna use in the simulate_alg function.

void reset_2 (void)
{
    int i,j,p,k;

    //We begin the batterie values of the devices
    for(i=0;i<num_dev;i++)
        batterie[i]=100000;

    for(i=0;i<num_dev;i++)
        array_graph[i][0]=100;
}
```

```
life=0;

//We create the array_graph_0, just to draw a line always
//with value 0, so we can see the graph between
//100 and 0
for(i=0;i<life;i++)
    array_graph_0[i]=0;
}

/*****FUNCION SIMULATE ALGORITHM*****/
//Function to simulate the result that we obtained in the run_alg
//function. With this function, we will try all the different
//configurations, and we will build the arrays to print the chart
//of the batterie evolution along time of each device

void simulate_alg(void)
{
    int i,j,p,k;

    //We print the life of everyconfiguration
    for(i=0;i<ind_conf_1;i++)
    {
        if (configurations_1[used_conf_1[i]])
        {

simulate_graph_1(used_conf_1[i],configurations_1[used_conf_1[i]]
);
        }
    }

    for(i=0;i<ind_conf_2;i++)
    {
        if (configurations_2[used_conf_2_1[i]][used_conf_2_2[i]])
        {

simulate_graph_2(used_conf_2_1[i],used_conf_2_2[i],configuration
s_2[used_conf_2_1[i]][used_conf_2_2[i]]);
        }
    }

    //printf("\n");
    for(i=0;i<ind_conf_3;i++)
```

```
    {
        if
(configuration_3[used_conf_3_1[i]][used_conf_3_2[i]][used_conf_3_3[i]]
)
        {
            //printf("\n\n");
            //printf("\n\tMaster 1: %d - Master 2: %d - Master 3:
%d :
%f",used_conf_3_1[i],used_conf_3_2[i],used_conf_3_3[i],configuration_3
[used_conf_3_1[i]][used_conf_3_2[i]][used_conf_3_3[i]]);
            //printf("\n");

            simulate_graph_3(used_conf_3_1[i],used_conf_3_2[i],used_conf_3_3
[i],configuration_3[used_conf_3_1[i]][used_conf_3_2[i]][used_conf_3_3[
i]]);
        }
    }
}

/*****FUNCION RESET 3*****/
//Function to create some arrays to print the lives of the
//different scatternets

void reset_3(void)
{
    //Array to print the lives of the scatternets
    //We create the arrays to print the second graph, which
    //about the life of the scatternet, without using any
    //algorithm, with algorithm and with algorithm and change of
    //configuration
    array_2_alg_y[0]=0;
    array_2_alg_y[1]=life_no_change;
    array_2_alg_x[0]=4;
    array_2_alg_x[1]=4;

    array_2_alg_change_y[0]=0;
    array_2_alg_change_y[1]=life;
    array_2_alg_change_x[0]=6;
    array_2_alg_change_x[1]=6;

    //We create some arrays so, the graphs are nice when we print
    array_2_1_y[0]=0;
    array_2_1_y[1]=life+3000;
    array_2_1_x[0]=0;
    array_2_1_x[1]=0;
    array_2_0_y[0]=0;
    array_2_0_y[1]=life+3000;
    array_2_0_x[0]=8;
    array_2_0_x[1]=8;
}
}
```

```
//*****FUNCTION INSERT*****
//Function used to connect devices to a piconet. With this function
//we will connect the device "k" to the piconet "piconet", where
//the master is the device "i", with a "value". This value will
//number of piconet (1 piconet:0-9, 2 piconet:10-19, 3 piconet:20-29)
//and the number will depend on the time of assignment: first, second
//third...

void insert(int piconet,int k,int i,int value)
{
    int a;

    connect[i][k]=1;
    connect[k][i]=1;
    taken[k]=value; //We put it like chosen in piconet A (1=piconet
A)
    num_dev_pic[piconet]=num_dev_pic[piconet]+1;
    //We increase the number of devices in the piconet
    //We add the tx_power of the piconet

    spend[piconet]=spend[piconet]+tx_power[i][k];
    //conf=conf+dist[i][k];
}

//*****FUNCTION LOOK*****
//Function used to look for a device with a determined "value"
//so we can connect it to a piconet where the master is the
//device "i", and the masters of the others device are "j" and "p".

//If there are several devices in the wanted state, we will take
//that one which saves more battery, so that one which is "closest"
//to the wanted master than to the rest

int look(int i, int j, int p, int value)
{
    double max=-100,diff;
    int l=0,k;
    found=0;

    for(k=0;k<num_dev;k++)
    {

        if (num_masters==2)
        {
            if(taken[k]==value)
            {
                found=1;
                diff=(power[i][k]-power[j][k]);

                //We select the best
                if(diff>max)
                {
                    max=diff;
                    l=k;
                }
            }
        }
    }
}
```

```
    }
}

else if(num_masters==3)
{
    if(taken[k]==value)
    {
        found=1;
        diff=(power[i][k]-(power[j][k]+power[p][k]));

        //We select the best
        if(diff>max)
        {
            max=diff;
            l=k;
        }
    }
}

}

//We return the number of the device found
return(l);
}
```

```
//*****FUNCTION TURN*****
//Function used to know which master must choose a device for its
//It will choose that master whose piconet is not full yet, and that
//one which has least power consumption, which will not depend on
//the number of devices of the piconet, but the medium amount of
//consumption of the piconet
```

```
void turn (void)
{
    int k,l=0;

    double min,temp;

    min=2000000000;

    for(k=0;k<num_masters;k++)
    {
        if(num_dev_pic[k]<8)
        {
            temp=(spend[k]/(num_dev_pic[k]-1));
            if(temp<min)
            {
                min=temp;
                l=k;
            }
        }
    }
}
```

```
    }
  }

  turno=1;
}

/*****FUNCTION ORDER*****/
//Function to order the devices by the total received power

void order(void)
{

  int j,k;
  double temp;
  double array[20];

  for(j=0;j<num_dev;j++)
    array[j]=power[j][num_dev];

  //We order the total power received
  for(k=1;k<=num_dev-1;k++)
  {
    j=0;

    while(j<num_dev-k)
    {
      if(array[j]<array[j+1])
      {
        temp=array[j];
        array[j]=array[j+1];
        array[j+1]=temp;
      }
      j=j+1;
    }
  }

  //In ord we will keep a list with the devices ordered
  for (j=0;j<num_dev;j++)
  {
    for (k=0;k<num_dev;k++)
    {
      if(array[j]==power[k][num_dev])
        ord[j]=k;
    }
  }

  //We print the total power received ordered descendly
  //for(j=0;j<num_dev;j++)
```

```
        //printf("\nRSSI %d: %f, correspondiente al %d Bluetooth",
j,array[j],ord[j]);
}
```

3.4 definitions algorithm.h

```
//*****DEFINITIONS_ALGORITHM.H*****
```

```
//In this file we have the definitions of the functions and variables
//we will use to create and simulate our scatternet with our algorithm
```

```
//*****
```

```
void main_2(void);
void calc_distances(void);
void calc_rx_power(void);
void calc_tx_power(void);
void reset_1(void);
void reset_2(void);
void reset_3(void);
void run_alg(void);
void simulate_alg(void);
void piconet_1(void);
void piconet_2(void);
void piconet_3(void);

void order(void);
void masters_2(void);
void masters_3(void);
void masters_3_2(void);

void insert(int piconet,int k,int i,int value) ;
int look(int i, int j, int p, int value) ;
int zero_assign (int i, int j);
void zero_assign_3 (int i, int j,int p);
void first_assign (int i, int j);
void first_assign_3 (int i, int j, int p);

void second_assign (int i, int j);
void second_assign_3 (int i, int j,int p);
void third_assign (int i, int j,int bridge);
void third_assign_3 (int i, int j,int p);
void fourth_assign (int i, int j);
void fourth_assign_3 (int i, int j,int p);
void print_configurations(int i, int j);
void join_masters(int pic_A,int pic_B,int pic_C,int i, int j, int p,int
m);
void take_master(int pic, int i, int j);
void find_bridge(int i, int j, int p, int pic,int used);
```

```
void turn (void);
void turn_2 (void);
int calc_consume_1 (int master,double max_bat);
int calc_consume_2 (int master_A,int master_B,double max_bat);
int calc_consume_3 (int master_A,int master_B,int master_C,double
max_bat);
int calc_consume_3_2(int master_A,int master_B,int master_C,double
max_bat);
void simulate_graph_1(int master_A, int life_conf);
void simulate_graph_2(int master_A, int master_B,int life_conf);
void simulate_graph_3(int master_A, int master_B, int master_C,int
life_conf);
void calc_consume_graph_1(int life_conf);
void calc_consume_graph_2(int life_conf);
void calc_consume_graph_3(int life_conf);
int calc_consume_1_no_change (int master,double max_bat);
int calc_consume_2_no_change (int master_A,int master_B,double
max_bat);
int calc_consume_3_no_change (int master_A,int master_B,int
master_C,double max_bat);
void print_arrays(void);

int num_dev, max,ack_2pics,
ack_3pics,ack_3pics2,ack_4pics,num_dev_pic[5],found,num_masters,turno;
int nok,num_def[5];
int coord_x[40], coord_y[40];
int ord[40]; //In ord we have the devices
ordered by the total RSSI
int see[40][40]; //In see we have the devices every device can
discover
double dist[40][40]; //In dist we have the distances between the
devices
double power[40][41]; //In power we have the rx power for each device
from the rest
double tx_power[40][40];
double conf; //In conf we have the value for each configuration
double conf_3[40][40][40];
int taken[40]; //In taken we write if one device is as a
slave in piconet A (1), piconet B (2)
//as master in pic A (3),as master
in pic B (5), or in no piconet (0)
double spend[5];

int master[5];
int final[40],final_2[40];
double min;
double batterie[40],connect[40][40];
int life,life_no_change; //Life of the scatternet
int dead,dead_2; //To know if the scatternet life has
expired
int pic_1[20],pic_2[20], pic_3[20],pic_4[20]; //To organize the
piconets
int def_pic_1[20],def_pic_2[20], def_pic_3[20],def_pic_4[20];
int ind_1,ind_2,ind_3,ind_4; //Indexes for the pic_1,pic_2, and pic_3
arrays
double def_medium_batterie;
```

```
int lifetime;
int num_dev_1,num_dev_2,num_dev_3,num_dev_4,prueba;
double configurations_1[8]; //Here we safe the total life of a
scatternet with every master and one piconet
double configurations_2[15][15]; //Here we safe the total life of a
scatternet with one every master with tow piconets
double configurations_3[22][22][22]; //Here we safe the total life of a
scatternet with one every master with three piconets
int used_conf_1[20],
used_conf_2_1[20],used_conf_2_2[20],used_conf_3_1[20],used_conf_3_2[20]
,used_conf_3_3[20];
int ind_conf_1,ind_conf_2,ind_conf_3;
int min_3[3],ok_3[3],nok_3[3],temp_3[3];
double dist_masters[3], dist_slaves[3];

//Variables for the zero_assign_3 function
double array_graph[25][300000],array_graph_random[25][300000],
array_graph_no_change[25][300000],array_graph_0[300000];
double array_batterie[25][1000];
int array_time[1000], number_points;
double bound_master,bound_dead;
int dev_plot,dev_conf;
int change,change_2;
int
read_device;array_2_normal_x[2],array_2_normal_y[2],array_2_alg_y[2],ar
ray_2_alg_x[2],array_2_alg_change_y[2],array_2_alg_change_x[2],array_2_
0_x[2],array_2_0_y[2],array_2_1_x[2],array_2_1_y[2];
int array_map_x[2],array_map_y[2];

int def_alg_pic_1[50][8],def_alg_pic_2[50][8],def_alg_pic_3[50][8];
int
coord_x_alg_pic_1[50][8],coord_x_alg_pic_2[50][8],coord_x_alg_pic_3[50]
[8];
int
coord_y_alg_pic_1[50][8],coord_y_alg_pic_2[50][8],coord_y_alg_pic_3[50]
[8];
int coord_x_bridge_1_2[50][8], coord_y_bridge_1_2[50][8];
int coord_x_bridge_1_3[50][8], coord_y_bridge_1_3[50][8];
int coord_x_bridge_2_3[50][8], coord_y_bridge_2_3[50][8];
int bridge_1_2[50],bridge_1_3[50],bridge_2_3[50];

int num_alg_def_1[50],num_alg_def_2[50],num_alg_def_3[50];
double fact;

//for the caption (leyenda)
int legend_x_master[3], legend_y_master[3],legend_x_slave[3],
legend_y_slave[3],legend_x_slave_bridge[3], legend_y_slave_bridge[3];
int array_legend_x[3],array_legend_y[3];
```

3.5 2_piconet.h

```
//*****2_PICONET.H*****  
  
//In this file we will run our algorithm to make the best scatternet  
//with two piconets,so we can prolong the life of the scatternet as  
//The chosen configuration will be that one that saves more battery and  
//prolong the life of the scatternet  
  
//One device will be able to be master only if its battery left is at  
//least a "bound_master"% of the medium amount of battery of all the  
//This "bound_master" has not a fixed value, and it variates between 85  
//and 40 %, and its value is automatically chosen in such way that  
//that the life of the scatternet is as long as possible.  
  
//We will simulate the behavior of the chosen configuration, and we  
//two life times: for the first time we will calculate the life time  
//till one device is without battery (normally onw master) (this will  
//time of the configuration without change), and for the rest of  
//configurations we will calculate the life time till the battery left  
//is under the "bound_dead" value, which variates between 40% and 22%  
//of the medium amount of battery of all the devices, or till one  
//is without battery, which would be the dead of the scatternet.  
  
//*****  
  
/*****FUNCION 2_PICONET*****/  
//With this function we will select the best configuration with two  
//To do that we will try all the possible combinations of masters: 1  
//1 and 3,... and we will select the best one  
  
//The configuration will be a valid one if between both masters, all  
//devices can be connected. One device will be able to be a master only  
//its battery left is at least is the "bound_master" of the medium  
//battery of all the devices  
  
//If there is no way to find a suitable configuration, we will make a  
//scatternet with three piconets  
  
void piconet_2(void)  
{  
  
    int k=0,b,vecas;  
  
    lifetime=0;
```

```
ack_2pics=0;
num_masters=2;
bound_master=0.85;
bound_dead=0.4;
veces=0;

//We look for a configuration done with 2 piconets.
//The masters must have some batterie left,
//at least bound_master value; once we find the masters, they
//will work till the batterie of
//one of them reaches the bound_dead value. If we dont find two
//available masters, we will try
//to make the configuration with three masters.

do
{
    bound_master=bound_master-0.05;
    bound_dead=bound_dead-0.02;
    veces++;
    masters_2();
}
while((ack_2pics==0)&&(bound_master>0.4));

//We have found two available masters, so we can make the
//scatternet with 2 piconets
if(ack_2pics==1) //It's been possible with 2 piconets
{

    //We will keep the order of the configurations. If the
    //configuration has not been used yet
    //we will assign a number to it
    if(configurations_2[master[0]][master[1]]==0)
    {
        used_conf_2_1[ind_conf_2]=master[0];
        used_conf_2_2[ind_conf_2]=master[1];
        ind_conf_2++;
    }

    change_2++;

    //If it is the first configuration, we save the life of the
    // scatternet with this configuration
    //and reset again all the values so it can work again with
    //the change of configuration
    //so we can calculate the life time of the first
    //configuration that the algorithm
    //will choose, so we get the life time without changing
    //configuration
    if(change_2==1)
    {
```

```
lifetime=calc_consume_2_no_change(master[0],master[1],def_medium
_batterie);

        //We begin the batterie values of the devices
        for(k=0;k<num_dev;k++)
            batterie[k]=100000;
        lifetime=0;
    }

    //We calculate the life of this configuration
    lifetime=calc_consume_2(master[0],master[1],def_medium_batterie)
;

    configurations_2[master[0]][master[1]]=configurations_2[master[0]
][master[1]]+lifetime;

}

else
    piconet_3(); //We will try with three piconets
}

/*****FUNCTION MASTERS_2*****/
//Function to study the different configurations of the scatternet made
// with two piconets

//With every couple of masters we will run our algorithm. We will
//create a scatternet (if possible) with every couple of masters, and
//the scatternet with lowest battery consumption will be the chosen
//one. One device can only be master if its battery is higher than
//a "bound master" value

void masters_2(void)
{

    int i,j,m,k,a,l,bridge;
    double temp,max_pow,medium_batterie;
    min=1000;
    ack_2pics=0;

    //We try all the possible configurations, and so we try with all
    //possible
```

```
//couples of devices working as masters
for (i=0;i<num_dev;i++) //for 1
{
    for(j=0;j<num_dev;j++) //for 2
    {

        //Master A: Bluetooth(ord(i))
        //Master B: Bluetooth(ord(j))

        //We initialize the pic_1 and pic_2 arrays
        for (m=0;m<12;m++)
        {
            pic_1[m]=0;
            pic_2[m]=0;
        }

        //In pic_1 and pic_2 we will keep the devices of each piconet:
        //the first device will be the master and the rest the slaves
        //If both piconets are connected using a bridge slave, this
        //bridge will be the second device in pic_1 and pic_2 arrays

        //ind_1 and ind_2 are just the indexes for the arrays
        ind_1=0;
        ind_2=0;
        pic_1[ind_1]=i;
        pic_2[ind_2]=j;

        max=0,lifetime=0;
        max_pow=-100000,medium_batterie=0;
        bridge=0;

        //We calculate the medium amount of battery in each device
        for(m=0;m<num_dev;m++)
            medium_batterie=medium_batterie+batterie[m];

        //In medium_batterie we will keep the medium amount of battery
        //of all the devices
        medium_batterie=medium_batterie/num_dev;

        //We initialize the spend and num_dev_pics arrays
        for(k=0;k<5;k++)
        {
            spend[k]=0;
            num_dev_pic[k]=1;
        }

        conf=0;
    }
}
```

```
//We check that master A and master B are not the
// same device
if (i!=j)
{

    a=0;

    //First: check that we can see all the devices
    // with the two masters
    for(k=0;k<num_dev;k++)
        a=a+(see[i][k]||see[j][k]);

    if (a!=num_dev) //Not valid: we don't see all
                    // the slaves
    {
        conf=0;
    }

//If we can connect all the devices and the masters have the minimum
//battery we will go on with the algorithm; if not we will try with
//two others masters

    else
if((a==num_dev)&&((batterie[i])>(bound_master*medium_batterie))&&((batt
erie[j])>(bound_master*medium_batterie))) //We can see all the devices
    {

        ack_2pics=1; //We can make two piconets

//Now we have to assigne the devices

//We initialize the value of taken, because no devices have been yet
//connected to the masters
    for(k=0;k<num_dev;k++)
        taken[k]=0;

        //The masters are marked as taken (its
        //name is 08,18,28,...)
        taken[i]=8 ;
        taken[j]=18;

//ZERO: bridge assign. We will look for a device that can be used as a
//bridge between both masters. If there's no such a device, we will
//just connect both masters.If we use a bridge_slave, "bridge" will
//value 1; if not will value 0
        bridge=zero_assign(i,j);
```

```
//FIRST: we assigne the devices that are only seen by one master
                first_assign(i,j);

//SECOND: time to assign those devices seen by both masters, but that
//are seen with much more power with one than with the other
                second_assign (i,j);

//THIRD: time to assign the rest of devices. These will be those who
//are near of both masters. Each master will pick a new device when its
//tx_power is less than the tx_power of the other piconet, and while the
//number of devices in its piconet is less than 7 (with itself 8)

                if((num_dev_pic[0]+num_dev_pic[1])<(num_dev+bridge)&&(num_dev_pi
c[0]<=8)&&(num_dev_pic[1]<=8))
                {
                        third_assign (i,j,bridge);

//FOURTH: if it is not possible to find a slave bridge, we will connect
//both masters

                                if(bridge==0)
                                        fourth_assign (i,j);

//In conf we keep the value of the configuration.
//The configuration with the lowest value will be the best
//and chosen configuration
                                conf=spend[0]+spend[1];

                                temp=conf;

//We save the best configuration
                                if (temp<min)
                                {
                                        min=temp;
                                        master[0]=i;
                                        master[1]=j;
                                        for(k=0;k<num_dev;k++)
                                        {
                                                if(taken[k]<10)
                                                        final[k]=1;
                                                else
                                                        final[k]=2;
                                        }

//We save the best configuration of pic_1 and pic_2
                                for(k=0;k<num_dev_pic[0];k++)
                                        def_pic_1[k]=pic_1[k];
                                for(k=0;k<num_dev_pic[1];k++)
                                        def_pic_2[k]=pic_2[k];

def_medium_batterie=medium_batterie;
```

```
num_dev_1=num_dev_pic[0];
num_dev_2=num_dev_pic[1];

    }

    }
} //end elsif vemos todos los dispositivos

    } // end if 1
} //end for 2

} // end for 1

}

/*****FUNCTION ZERO ASSIGN*****/
//With this function we try to connect both piconets using a bridge e
//This bridge slave will only be valid if using it as the bridge,
//the battery consumption is lower than if we connect bothh masters

//If there are several valid bridge-slaves, we will choose the one
//that saves more batteries

int zero_assign (int i, int j)
{
    double dist_masters,dist_mas1,dist_mas2,temp;
    int k,ok,min;

    temp=1000;
    ok=0;

    //We keep the distance between both masters in "dist_masters"
    dist_masters=dist[i][j];

    //We check the rest of devices, and see if we find a valid
    //bridge-slave. If we find several ones, we will take the best
    //one
    for(k=0;k<num_dev;k++)
    {
        if ((k!=i)&&(k!=j))
        {
            //Distances between every master and the slave we are
```

```
//checking
dist_mas1=dist[i][k];
dist_mas2=dist[j][k];

//If the distance between the masters and the slave,
// is lower than the distance between both masters,
//the slave is a valid
//bridge-slave

if((dist_mas1<=dist_masters) &&
(dist_mas2<=dist_masters)) //If we find a valuable slave bridge
{
    ok=1;

    //If there are several bridge-slaves we take
    //the best one
    if((dist_mas1+dist_mas2)<temp)
    {
        temp=(dist_mas1+dist_mas2);
        min=k;
    }
}

}

if (ok) //If we have found a bridge slave, we will insert it
//in both piconets
{
    //We insert in piconet 1, with value 7
    taken[min]=7; //We put it like chosen in piconet A
    num_dev_pic[0]++; //We increase the number of devices in the
//piconet
    //We add the tx_power of the piconet
    spend[0]=spend[0]+(tx_power[i][min]);
    ind_1++;
    pic_1[ind_1]=min;

    //We insert in piconet 2, with value 17
    taken[min]=17; //We put it like chosen in piconet A
    num_dev_pic[1]++; //We increase the number of devices in the
//piconet
    //We add the tx_power of the piconet
    spend[1]=spend[1]+(tx_power[j][min]);
    ind_2++;
    pic_2[ind_2]=min;

    //We save the bridge coordinates to print them in the map
    //coord_x_bridge_1_2[=coord_x[min];
    //coord_y_bridge_1_2=coord_y[min];

}

return(ok);
```

```
}

/*****FUNCTION FIRST ASSIGN*****/
//With this function, we will connect to each piconet, those devices
//that are only seen by one master. At the end of this function
//the devices wich have not been connected yet to any piconet, will
//be a bunch of devices that are seen by both masters

void first_assign (int i, int j)
{
    int k;

    //We check all the devices that have not been connected
    for(k=0;k<num_dev;k++)
    {
        if (taken[k]==0)
        {
            //If only Master A is able to see the slave
            if((see[i][k]==1) && (see[j][k]==0)) //Only master A
            {
                insert(0,k,i,1);
                ind_1++;
                pic_1[ind_1]=k;
            }

            //If only Master B is able to see the slave
            if((see[i][k]==0) && (see[j][k]==1)) //Only master B
            {
                insert(1,k,j,1);
                ind_2++;
                pic_2[ind_2]=k;
            }
        }
    }
}

/*****FUNCTION SECOND ASSIGN*****/
//Function to assign those devices that are seen by both master, but
//seen with much more power with one than with the other

//The function works with a do-while loop. Every time this loop is run
//we try to assign those devices that are seen with much more power
//by one master than by the other, but also everytime it is run, we
//decrease the value of "dif", so we assign first the most critical
//devices, and then we assign those who are not so critical. At the end
//we will have a bunch of devices that are seen by both masters, and
//more or less with the same power, so those devices wont be really
//critical
```

```
void second_assign (int i, int j)
{
    int k;
    double diff,dif;

    //Initial value of dif
    dif=10;

    //Everytime we run the do-while loop, the value of dif is lower
    do
    {
        for(k=0;k<num_dev;k++)
        {
            if(taken[k]==0)
            {
                diff=(dist[i][k]-dist[j][k]);

                //If the difference of distance is important we
                //assign the
                //device to the nearest master
                if (fabs(diff)>dif)
                {
                    if ((diff<0)&&(num_dev_pic[0]<8))
                    {
                        insert(0,k,i,2);
                        ind_1++;
                        pic_1[ind_1]=k;
                    }

                    if ((diff>=0)&&(num_dev_pic[1]<8))
                    {
                        insert(1,k,j,12);
                        ind_2++;
                        pic_2[ind_2]=k;
                    }
                }
            }
        }

        //We decrease the value of dif, so in the next time, we try
        //to assign those devices where the difference is not so big
        dif=dif-0.5;
    }
    while(dif>=1);
}
```

```
/******FUNCTION THIRD ASSIGN******/
//Before executing this function, the devices that havent been onnected
//yet are those ones, whore are seen by both masters, more or less with
//the same power, so it is not something critical, if they are assigned
//to one piconet or to the other. So, it will the piconets who will
```

```
//choose device: the piconet the least amount of medium received power
//will be the one in choosing

void third_assign (int i, int j,int bridge)
{

    int k,l;

    //We execute this loop, till we have assigned all the devices
    //to one piconet

    do
    {
        //With turn function, we will know which piconet will
        //choose: it will be that one with the least medium amount
        //of power received
        turn();

        //printf("/nTurno:%d",turno);
        //The piconet A chooses
        switch (turno)
        {
            case 0:
            {

                //With "look" we look for a free device, and if
                //there are several, we will choose the best ne
                l=look(i,j,0,0);
                if(found==1)
                {
                    insert(0,l,i,3);
                    ind_1++;
                    pic_1[ind_1]=1;
                }
                break;
            }

            case 1:
            {

                //With "look" we look for a free device, and if there are
                //several, we will choose the best one
                l=look(j,i,0,0);
                if(found==1)
                {
                    insert(1,l,j,13);
                    ind_2++;
                    pic_2[ind_2]=1;
                }
                break;
            }

            default: break;
        }
    }
}
```

```
    }
    while ((num_dev_pic[0]+num_dev_pic[1])!=(num_dev+bridge));
}

/*****FUNCTION FOURTH ASSIGN*****/
//We use this function to connect the two masters: one master
//will work as a slave in the other piconet.
//We will only connect both master if we are not using a bridge-slave

void fourth_assign (int i, int j)
{
    //With turn_2, we say which piconet will take the other master
    //as a slave: it will be that one will the lowest number of devices
    turn_2();

    switch (turno)          //The piconet A chooses
    {
        case 0:
        {
            taken[j]++; //We put it like chosen in piconet A
            num_dev_pic[0];
            spend[0]=spend[0]+(tx_power[i][j]);
            ind_1++;
            pic_1[ind_1]=j;
            break;
        }
        //The piconet B chooses
        case 1:
        {
            taken[i]++; //We put it like chosen in piconet B
            num_dev_pic[1];
            spend[1]=spend[1]+(tx_power[j][i]);
            ind_2++;
            pic_2[ind_2]=i;
            break;
        }
        default: break;
    }
}

/*****FUNCTION TURN 2*****/
//We use this function when we connect both piconets, connecting both
//In this function we will choose the piconet that will take the other
//as a slave. It will be that one with the least number of devices
```

```
void turn_2 (void)
{
    int k;

    double min,temp;

    if (num_dev_pic[0]>num_dev_pic[1])
        turno=1;
    else if(num_dev_pic[0]<num_dev_pic[1])
        turno=0;
    else if (num_dev_pic[0]==num_dev_pic[1])
        turn;

}

/***** CONSUME BATTERIES SIMULATION *****/
//Function to simulate the consumption of battery in the configuration
//This configuration will live till one master has less than the
//battery left, or because one device is with no more battery

//We consider all the devices with a initial value of 100000
//and depending on the time that they are used and the distance
//from the other device it will decrease faster or slower

int calc_consume_2 (int master_A,int master_B,double max_bat)
{
    int i,j,time,slot;
    time=0;
    dead=0;

    //This loop is repeated every unit time. In each unit time
    //the master and one slave will transmit (consume). To know
    //transmits in each unit time, we use the variable "slot", which
    //depend on the number of devices of the piconet and the moment

    do
    {

        //Piconet 1
        if((num_dev_1<=8)&&(num_dev_1>=2))
        {
            slot=time%(num_dev_1-1);
            //Consume of the master
            batterie[def_pic_1[0]]=batterie[def_pic_1[0]]-
            dist[def_pic_1[0]][def_pic_1[1+slot]];

            //Consume of the slaves
```

```
    batterie[def_pic_1[1+slot]]=batterie[def_pic_1[1+slot]]-
    dist[def_pic_1[0]][def_pic_1[1+slot]];
    }

    //Piconet 2
    if((num_dev_2<=8)&&(num_dev_2>=2))
    {
        slot=time%(num_dev_2-1);
        //Consume of the master
        batterie[def_pic_2[0]]=batterie[def_pic_2[0]]-
        dist[def_pic_2[0]][def_pic_2[1+slot]];

        //Consume of the slaves

        batterie[def_pic_2[1+slot]]=batterie[def_pic_2[1+slot]]-
        dist[def_pic_2[0]][def_pic_2[1+slot]];
    }

//We check is there is a slave with no more battery. When one device
//is without battery, the piconet will be "dead".

    for (i=0;i<num_dev;i++)
    {
        if (batterie[i]<=0)
            dead=1;
    }

    time++;

}
while
((batterie[master_A]>((bound_dead)*(max_bat)))&&(batterie[master_B]>((b
ound_dead)*(max_bat)))&&(dead==0));

    life=life+time;
    return time;
}

//***** CONSUME BATTERIES SIMULATION *****
//Function to simulate the consumption of battery in the configuration
//This configuration will live till one device is with no more battery

//*We consider all the devices with a initial value of 100000
//*and depending on the time that they are used and the distance
//*from the other device it will decrease faster or slower

int calc_consume_2_no_change (int master_A,int master_B,double max_bat)
```

```
{
    int i,j,time,slot;
    time=0;
    dead=0;

//This loop is repeated every unit time. In each unit time
//the master and one slave will transmit (consume). To know which slave
//transmits in each unit time, we use the variable "slot", which will
//depend on the number of devices of the piconet and the moment of time

    do
    {

//We check the number of devices of the piconet, and with that and
//"time" we get "slot", and so we know which slave is working now

        //Piconet 1
        if((num_dev_1<=8)&&(num_dev_1>=2))
        {
            slot=time%(num_dev_1-1);
            //Consume of the master
            batterie[def_pic_1[0]]=batterie[def_pic_1[0]]-
dist[def_pic_1[0]][def_pic_1[1+slot]];

            //Consume of the slaves

            batterie[def_pic_1[1+slot]]=batterie[def_pic_1[1+slot]]-
dist[def_pic_1[0]][def_pic_1[1+slot]];
        }

        //Piconet 2
        if((num_dev_2<=8)&&(num_dev_2>=2))
        {
            slot=time%(num_dev_2-1);
            //Consume of the master
            batterie[def_pic_2[0]]=batterie[def_pic_2[0]]-
dist[def_pic_2[0]][def_pic_2[1+slot]];

            //Consume of the slaves

            batterie[def_pic_2[1+slot]]=batterie[def_pic_2[1+slot]]-
dist[def_pic_2[0]][def_pic_2[1+slot]];
        }

//We check is there is a slave with no more battery. When one device
//is without battery, the piconet will be "dead".

        for (i=0;i<num_dev;i++)
        {
            if (batterie[i]<=0)
                dead=1;
        }

        time++;
    }
}
```

```
//We keep in array_graph_random the value of the batterie left in each
// device in every moment of time. Later we will use this points to
//draw a graph
    for(i=0;i<num_dev;i++)
        array_graph_no_change[i][time]=(batterie[i]/1000);

    }
    while (dead==0);

    life_no_change=life_no_change+time;
    return time;}
```

3.6 3 simulate.h

```
/******3_SIMULATE.H*****
//In this file we will simulate the behavior of a scatternet made with
//We will indicate who the master are, who the slaves are and the life
//so we can simulate the evolution of the batteries of each device
//For each configuration, we will also keep the coordinates of each
//to make the representation in the graphical interface in the space

/******
//*****FUNCTION SIMULATE GRAPH 3 *****
//We use this function to simulate the way the scatternet will work
//We will tell who the masters are and how long will this configuration
//The function will apply the algorithm to make the scatternet,
//so we can get see the batterie left in each Bluetooth after
//working this configuration. Then with this, we will make the graphs

void simulate_graph_3(int master_A, int master_B,int master_C,int
life_conf)
{
    int i,j,p,m,k,a,l;
    double temp,max_pow,medium_batterie;
    min=1000;

    i=master_A;
    j=master_B;
    p=master_C;

    //Master A: Bluetooth(i)
    //Master B: Bluetooth(j)
    //Master C: Bluetooth(p)

    //We initialize the pic_1 and pic_2 arrays
    for (m=0;m<20;m++)
    {
        pic_1[m]=0;
        pic_2[m]=0;
    }
}
```

```
        pic_3[m]=0;
    }

    ind_1=0;
    ind_2=0;
    ind_3=0;
    pic_1[ind_1]=i;
    pic_2[ind_2]=j;
    pic_3[ind_3]=p;

    max=0,lifetime=0;
    max_pow=-100000,medium_batterie=0;

    num_masters=3;
    for (k=0;k<num_masters;k++)
    {
        num_dev_pic[k]=1;
        spend[k]=0;
    }

    conf=0;

//Now we have to assigne the devices
//We initialize the value of taken, because no devices have been yet
//connected to the masters
    for(k=0;k<num_dev;k++)
        taken[k]=0;

//The masters are marked as taken
taken[i]=8;    //Master of Pic A
taken[j]=18;   //Master of Pic B
taken[p]=28;   //Master of Pic C
nok=0;

//ZERO: bridge assign. We will look for a device that can be used as a
//bridge between both masters. If there's no such a device, we will
//just connect both masters.
    zero_assign_3(i,j,p);

//FIRST: we assigne the devices that are only seen by one master
    first_assign_3(i,j,p);

//SECOND: time to assign those devices seen by both masters, but that
//are seen with much more power with one than with the other
    second_assign_3 (i,j,p);
```

```
//THIRD: time to assign the rest of devices. These will be those who
//are near of both masters. Each master will pick a new device when its
//tx_power is less than the tx_power of the other piconet, and while the
//number of devices in its piconet is less than 7 (with itself 8)

    if((num_dev_pic[0]+num_dev_pic[1]+num_dev_pic[2])<(num_dev+2))
        third_assign_3 (i,j,p);

//We calculate the batterie left in each device after a life_conf with
//this configuration
    calc_consume_graph_3(life_conf);

    change++;

//We keep the configuration, so we can print it in the interface (space
//map): we keep the coordinates of this random configuration

//In def_random_pic_1, we keep the piconet configuration: the first
//device will be the master, and the rest the slaves. In
//coord_x_random_pic we keep the coordinates of the configuration.

//We add one, because the indexes in the graphic representation is
//different from the one we use to process (first device:1, first
//device:0)
    for(k=0;k<num_dev_pic[0];k++)
    {
        def_alg_pic_1[change-1][k]=pic_1[k]+1;
        coord_x_alg_pic_1[change-1][k]=coord_x[pic_1[k]];
        coord_y_alg_pic_1[change-1][k]=coord_y[pic_1[k]];
    }

    for(k=0;k<num_dev_pic[1];k++)
    {
        def_alg_pic_2[change-1][k]=pic_2[k]+1;
        coord_x_alg_pic_2[change-1][k]=coord_x[pic_2[k]];
        coord_y_alg_pic_2[change-1][k]=coord_y[pic_2[k]];
    }

    for(k=0;k<num_dev_pic[2];k++)
    {
        def_alg_pic_3[change-1][k]=pic_3[k]+1;
        coord_x_alg_pic_3[change-1][k]=coord_x[pic_3[k]];
        coord_y_alg_pic_3[change-1][k]=coord_y[pic_3[k]];
    }

//In num_rand_def we keep the number of devices of the piconet in this
//configuration
    num_alg_def_1[change-1]=num_dev_pic[0];
    num_alg_def_2[change-1]=num_dev_pic[1];
    num_alg_def_3[change-1]=num_dev_pic[2];
```

```
//We look for a bridge_slave, between master 1 and 2. If there is a
//bridge_slave we keep it in one array so we can keep it in the
//interface
    if((pic_1[1]==pic_2[1])||(pic_1[1]==pic_2[2]))
    {
        bridge_1_2[change-1]=1;
        coord_x_bridge_1_2[change-1][0]=coord_x[pic_1[1]];
        coord_y_bridge_1_2[change-1][0]=coord_y[pic_1[1]];
    }

    else if((pic_1[2]==pic_2[2])||(pic_1[2]==pic_2[1]))
    {
        bridge_1_2[change-1]=1;
        coord_x_bridge_1_2[change-1][0]=coord_x[pic_1[2]];
        coord_y_bridge_1_2[change-1][0]=coord_y[pic_1[2]];
    }

//We look for a bridge_slave, between master 2 and 3. If there is a
//bridge_slave we keep it in one array so we can keep it in the
//interface
    if((pic_2[1]==pic_3[1])||(pic_2[1]==pic_3[2]))
    {
        bridge_2_3[change-1]=1;
        coord_x_bridge_2_3[change-1][0]=coord_x[pic_2[1]];
        coord_y_bridge_2_3[change-1][0]=coord_y[pic_2[1]];
    }

    else if((pic_2[2]==pic_3[2])||(pic_2[2]==pic_3[1]))
    {
        bridge_2_3[change-1]=1;
        coord_x_bridge_2_3[change-1][0]=coord_x[pic_2[2]];
        coord_y_bridge_2_3[change-1][0]=coord_y[pic_2[2]];
    }

//We look for a bridge_slave, between master 1 and 3. If there is a
//bridge_slave we keep it in one array so we can keep it in the
//interface
    if((pic_1[1]==pic_3[1])||(pic_1[1]==pic_3[2]))
    {
        bridge_1_3[change-1]=1;
        coord_x_bridge_1_3[change-1][0]=coord_x[pic_1[1]];
        coord_y_bridge_1_3[change-1][0]=coord_y[pic_1[1]];
    }

    else if((pic_1[2]==pic_3[2])||(pic_1[2]==pic_3[1]))
    {
        bridge_1_3[change-1]=1;
        coord_x_bridge_1_3[change-1][0]=coord_x[pic_1[2]];
        coord_y_bridge_1_3[change-1][0]=coord_y[pic_1[2]];
    }
```

```
}
```

```
/****** FUNCTION CALC_CONSUME_GRAPH_2 *****  
//With this function, we calculate the batterie left in each device,  
//after running a configuration where master_A, and master_B are the  
//masters of the piconets, after a life_conf time
```

```
void calc_consume_graph_3(int life_conf)
```

```
{  
    int i,j,time,slot,tempo;  
    time=0;  
    dead=0;  
    tempo=0;
```

```
//This loop is repeated "life_conf" times. In each time  
//the master and one slave will transmit (consume). To know which slave  
//transmits in each unit time, we use the variable "slot", which will  
//depend on the number of devices of the piconet and the moment of time
```

```
    for(j=0;j<life_conf;j++)  
    {
```

```
//We check the number of devices of the piconet, and with that and  
//"time" we get "slot", and so we know which slave is working now
```

```
        //Piconet 1  
        if((num_dev_pic[0]<=8)&&(num_dev_pic[0]>=2))  
        {  
            slot=time%(num_dev_pic[0]-1);  
            //Consume of the master  
            batterie[pic_1[0]]=batterie[pic_1[0]]-  
dist[pic_1[0]][pic_1[1+slot]];  
  
            //Consume of the slaves  
            batterie[pic_1[1+slot]]=batterie[pic_1[1+slot]]-  
dist[pic_1[0]][pic_1[1+slot]];  
        }
```

```
        //Piconet 2  
        if((num_dev_pic[1]<=8)&&(num_dev_pic[1]>=2))  
        {  
            slot=time%(num_dev_pic[1]-1);  
            //Consume of the master  
            batterie[pic_2[0]]=batterie[pic_2[0]]-  
dist[pic_2[0]][pic_2[1+slot]];  
  
            //Consume of the slaves
```

```
        }
```

```
        batterie[pic_2[1+slot]]=batterie[pic_2[1+slot]]-
dist[pic_2[0]][pic_2[1+slot]];
    }

    //Piconet 3
    if((num_dev_pic[2]<=8)&&(num_dev_pic[2]>=2))
    {
        slot=time%(num_dev_pic[2]-1);
        //Consume of the master
        batterie[pic_3[0]]=batterie[pic_3[0]]-
dist[pic_3[0]][pic_3[1+slot]];

        //Consume of the slaves
        batterie[pic_3[1+slot]]=batterie[pic_3[1+slot]]-
dist[pic_3[0]][pic_3[1+slot]];
    }

    time++;
    life++;

//We keep in array_graph the batterie left in each device in each
//moment of time. we will take one point of 20, so we dont have a
//really big array
    for (i=0;i<num_dev;i++)
    {
        array_graph[i][life]=(batterie[i]/1000);
    }
}
}
```

3.7 main_random.h

```
//*****MAIN_RANDOM.H*****

//We will use this file to simulate a random behavior to make
//a scatternet. We will create a random scatternet and will simulate
//its lifetime and battery evolution along time, so we can compare
//later to the results we get with our algorithm
//*****

/*****FUNCION MAIN RANDOM*****/
//Depending on the number of devices, we will call one or other
//function to create our random scatternet

void main_random ()
{
    int i,j,p;

    //We reset some arrays and variables
    reset_random();

    //We call a function depending on the number of devices
    do
    {
        if (num_dev<=8)          //If we have less than 8 devices we
                                //will try just one piconet
            piconet_1_random();

            else if (num_dev<16 && num_dev>8)
//If we have 9-15 devices, we will try two piconets
                piconet_2_random();

            else if (num_dev<23 && num_dev>15)
//We will try three piconets
                piconet_3_random();
    }
```

```
    }
    while (random_ready==0);

    //We create the array to represent the life of the scatternet
    //with a normal behavior.
    array_2_normal_y[0]=0;
    array_2_normal_y[1]=life_random;
    array_2_normal_x[0]=2;
    array_2_normal_x[1]=2;

}    //End of main

/*****FUNCION RESET RANDOM*****/
//Function to reset some variables and arrays that we the
//functions that will make the scatternet will need

void reset_random(void)
{
    int i,j,p;

    dead=0,dead_2=0,life_random=0,random_ready=0;
    num_rand_def_1=num_rand_def_2=num_rand_def_3=0,num_rand_def_4=0;
    fact=1;

    //We beguin the batterie values of the devices
    for(i=0;i<num_dev;i++)
        batterie[i]=100000;

    //We check that the devices still have batteries, so they are
    //not dead
    for(i=0;i<num_dev;i++)
        if(batterie[i]<0)
            dead=1;

    //We reset the configurations arrays where we keep the lifetime
    //of the scatternet of
    //every different configuration
    for(i=0;i<8;i++)
    {

        def_random_pic_1[i]=0;
        def_random_pic_2[i]=0;
        def_random_pic_3[i]=0;
        def_random_pic_4[i]=0;
        coord_x_random_pic_1[i]=0;
        coord_x_random_pic_2[i]=0;
        coord_x_random_pic_3[i]=0;
        coord_y_random_pic_1[i]=0;
    }
}
```

```
        coord_y_random_pic_2[i]=0;
        coord_y_random_pic_3[i]=0;
    }

    //We reset the array_graph array
    for(i=0;i<num_dev;i++)
        for(j=0;j<100000;j++)
            array_graph_random[i][j]=0;

    for(i=0;i<num_dev;i++)
        array_graph_random[i][0]=100;
}
```

3.8 definitions_random.h

```
//*****DEFINITIONS_RANDOM.H*****

//In this file we have the definitions of the functions and variables
//that we will use to create and simulate a random scatternet

//*****

void main_random(void);
void reset_random(void);
void piconet_1_random(void);
void piconet_2_random(void);
void piconet_3_random(void);

void masters_2_random(void);
void masters_3_random(void);

int zero_assign_random (int i, int j);
int first_assign_random (int i, int j);
void first_assign_3_random (int i, int j, int p);
void first_assign_4_random (int i, int j, int p,int o);

void second_assign_random (int i, int j);
void second_assign_3_random (int i, int j,int p);
void second_assign_4_random (int i, int j,int p,int o);
int calc_consume_1_random (int master,double max_bat);
int calc_consume_2_random (int master_A,int master_B,double max_bat);
int calc_consume_3_random (int master_A,int master_B,int
master_C,double max_bat);

int life_random;

//We create some arrays to print the scatternet configuration in the
graph interface
int
def_random_pic_1[8],def_random_pic_2[8],def_random_pic_3[8],def_random_
pic_4[8];
```

```
int
coord_x_random_pic_1[8],coord_x_random_pic_2[8],coord_x_random_pic_3[8]
,coord_x_random_pic_4[8];
int
coord_y_random_pic_1[8],coord_y_random_pic_2[8],coord_y_random_pic_3[8]
,coord_y_random_pic_4[8];
int num_rand_def_1,num_rand_def_2,num_rand_def_3,num_rand_def_4;
int random_ready,cont_trys_2,cont_trys_3;

int coord_x_bridge_rand_1_2[8], coord_y_bridge_rand_1_2[8];
int coord_x_bridge_rand_1_3[8], coord_y_bridge_rand_1_3[8];
int coord_x_bridge_rand_2_3[8], coord_y_bridge_rand_2_3[8];
int bridge_rand_1_2,bridge_rand_1_3,bridge_rand_2_3;
```

3.9 1 piconet random.h

```
/******1_PICONET_RANDOM.H*****

//In this file we will simulate a normal behavior to get a piconet.
//Randomly we will choose the master and make the piconet. The piconet
//will be valid if with the chosen master all the slaves are
//discoverable, if not, we will try with another master. The number
//of attempts will be kept in "cont_trys". After 3 unsuccessfull attempts
//we will try a scatternet made with two piconets

//When the configuration is ready, we will simulate its time of life,
//so we can compare later to the time we get with our algorithm

/******

/*****FUNCION 1_PICONET_RANDOM*****/
//Function to get the master of the piconet
//The master will be chosen randomly, and to be a valid master
//it must be able to get connected to to all the slaves

void piconet_1_random(void)
{
    int i,j,a,b,lifetime,c,cont_trys;
    double max_pow,medium_batterie;

    max=0,b=0,lifetime=0,random_ready=0,cont_trys=0;
    max_pow=-100000,medium_batterie=0;
    num_masters=1;

    //We reset the connect array
    for(i=0;i<num_dev;i++)
        for(j=0;j<num_dev;j++)
            connect[i][j]=0;

    //We calculate the medium amount of battery in each device and
    // reset the piconet_1 array
    for(i=0;i<num_dev;i++)
```

```
{
    medium_batterie=medium_batterie+batterie[i];
    pic_1[i]=0;
}

medium_batterie=medium_batterie/num_dev;

//We select one device randomly. If it is possible to build the
// piconet with it, it
//will be chosen as master
do
{
    srand(time(0));
    i=rand()%(num_dev+1);
    cont_trys++;

    a=0;

    //We check the number of devices that the master can see
    for (j=0;j<num_dev;j++)
        a=a+see[i][j];

//If the master master can see all the devices it is valid master
//If not, we must look for another master or change the number of
//piconets
    if (a==(num_dev-1))
    {
//We organize the piconet. In piconet_1[0] we will keep the master, and
//in the rest we will keep the slaves (piconet_1[1], piconet_1[2],...
//piconet_1[num_dev-1])
        b=1;
        c=0;
        pic_1[0]=i;

        for (j=0;j<num_dev;j++)
        {
            if(j!=i)
            {
                if(c==0)
                    pic_1[j+1]=j;
                else
                    pic_1[j]=j;
            }

            else
                c++;
        }

//We keep the configuration, so we can print it in the interface (space
//map): we keep the coordinates of this random configuration

//In def_random_pic_1, we keep the piconet configuration: the first
//device will be the master, and the rest the slaves. In
//coord_x_random_pic we keep the coordinates of the configuration.
```

```
//We add one, because the indexes in the graphic representation is
//different from the one we use to process (first device:1, first
//device:0)
    for(j=0;j<num_dev;j++)
    {
        def_random_pic_1[j]=pic_1[j]+1;
        coord_x_random_pic_1[j]=coord_x[pic_1[j]];
        coord_y_random_pic_1[j]=coord_y[pic_1[j]];
    }

//In num_rand_def we keep the number of devices of the piconet in this
//configuration
    num_rand_def_1=num_dev;
    random_ready=1;

//We have to know which devices are connected. 1=connected; 0=not
//connected
    for(j=0;j<num_dev;j++)
    {
        if (j!=i)
        {
            connect[j][i]=1;
            connect[i][j]=1;
        }
    }

    //We calculate the life of this configuration
    lifetime=calc_consume_1_random(i,medium_batterie);

}

}

//The process will be repeated till we get a valid configuration
//or we repeat the process three times without success, and then
//we try to make two piconets
    while((random_ready==0)&&(cont_trys<3));

    if(random_ready==0)
        piconet_2_random;
}

//***** CALC CONSUME 1 RANDOM *****/
//Function to simulate the consumption of battery in the configuration

//*We consider all the devices with a initial value of 100000
//*and depending on the time that they are used and the distance
//*from the other device it will decrease faster or slower
```

```
int calc_consume_1_random (int master,double max_bat)
{
    int i,j,time,slot;
    time=0;
    dead=0;

    //This loop is repeated every unit time. In each unit time
    //the master and one slave will transmit (consume). To know which slave
    //transmits in each unit time, we use the variable "slot", which will
    //depend on the number of devices of the piconet and the moment of time
    do
    {

        //We check the number of devices of the piconet, and with that and
        // "time" we get "slot", and so we know which slave is working now
        if((num_dev<=8)&&(num_dev>=2))
        {
            slot=time%(num_dev-1);

            //Consume of the master
            batterie[pic_1[0]]=batterie[pic_1[0]]-
            dist[pic_1[0]][pic_1[1+slot]];

            //Consume of the slaves
            batterie[pic_1[1+slot]]=batterie[pic_1[1+slot]]-
            dist[pic_1[0]][pic_1[1+slot]];
        }

        //We check is there is a slave with no more battery. When one device
        //is without battery, the piconet will be "dead".
        for (i=0;i<num_dev;i++)
        {
            if (batterie[i]<=0)
                dead=1;
        }

        time++;

        //We keep in array_graph_random the value of the batterie left in each
        //device in every moment of time. Later we will use this points to draw
        //a graph
        for(i=0;i<num_dev;i++)
            array_graph_random[i][time]=(batterie[i]/1000);

    }
    while (dead==0);

    //We keep the life of the piconet
    life_random=life_random+time;
}
```

```
    return time;  
}
```

USER'S MANUAL

4.1 INTRODUCTION

This program has been developed to work with a maximum number of 22 devices, and so it can simulate the creation of scatternets made of a maximum of 3 piconets. It could have been done for a higher number of devices, but 22 is a high enough amount if we think about Bluetooth. Basically, the program will simulate a random scatternet configuration (simulating nowadays behavior) and two algorithm configurations (static and dynamic), and compare the results (life of the configurations, battery consumption of the devices) of all them.

The execution of the program will be controlled from a graphical user interface that can be seen in the Figure, divided into 8 different parts:

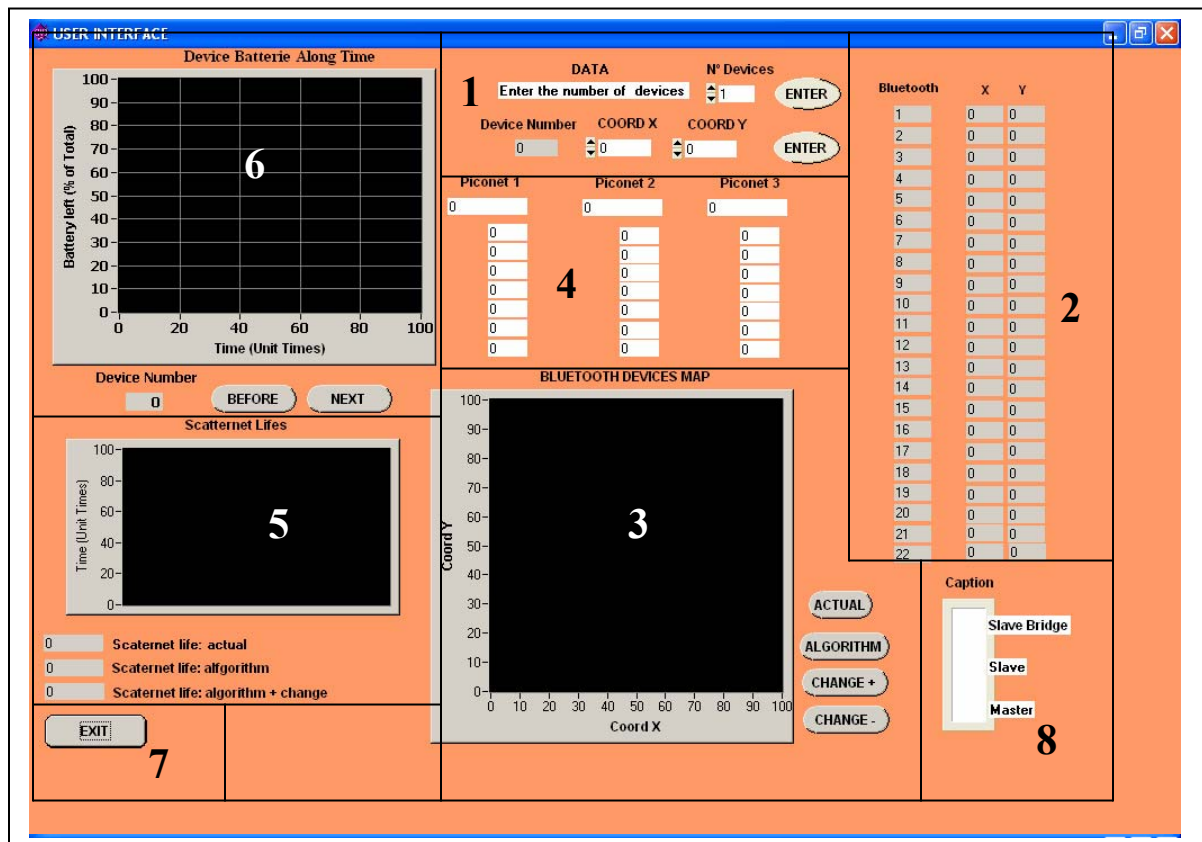


Figure 30: Graphical User Interface

4.2 Graphical User Interface

The different parts of the graphical interface are:

- 1. Data Entry Part:** in this part the user will enter the number of devices and their coordinates. The way it works is really easy: just write the number or the coordinates, and push Enter.

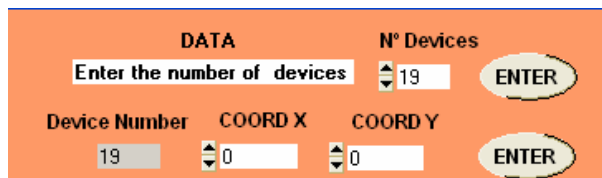
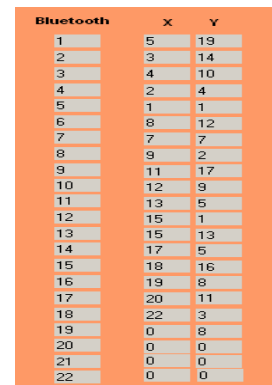


Figure 31: Data Entry Part



Bluetooth	X	Y
1	5	19
2	3	14
3	4	10
4	2	4
5	1	1
6	8	12
7	7	7
8	9	2
9	11	17
10	12	9
11	13	5
12	15	1
13	15	13
14	17	5
15	18	16
16	19	8
17	20	11
18	22	3
19	0	8
20	0	0
21	0	0
22	0	0

Figure 32: Bluetooth Devices Coordinates

- 2. Bluetooth Devices Coordinates Part:** this is just a place where the coordinates of every device will be written, as soon as the user enters them in the Data Entry Section.

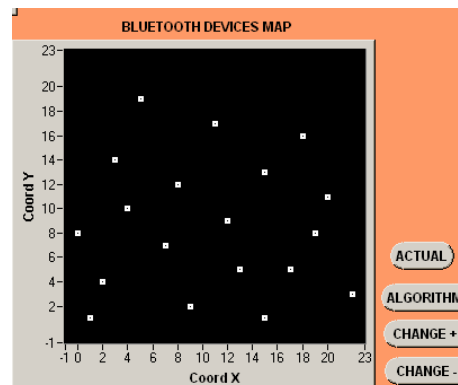


Figure 33: Initial Bluetooth Devices Map

3. Bluetooth Devices Map Part: this is a two-dimension space map for the Bluetooth devices. Here the user will have a spatial view, which is the best one to appreciate the differences between the several piconet/scatternet configurations:

- First of all, when the devices coordinates are entered, they will appear in the map in white color. This is the initial view, where the devices are free, and there is no piconet/scatternet made yet.

- Whenever it is pressed one of the buttons (**ACTUAL**, **ALGORITHM**, **CHANGE +**, **CHANGE -**), it will be seen a new view, where it can be distinguished the different elements of a piconet/scatternet of the configuration created:
 - To distinguish the devices of the piconets (if there are several), different colors for each one will be used: yellow, blue and red.
 - A Master will be represented by a square, a slave by an empty circle, and a bridge-slave by a full circle.

Pressing the different buttons, the configuration map will be changed:

- **ACTUAL:** pressing this button, it is printed the representation of the current (random) configuration.
- **ALGORITHM:** pressing this button, it is printed the representation of the static algorithm, which is the same one that the first configuration of the dynamic algorithm.
- **CHANGE +:** with this button, it is printed the representation of the next configuration of the dynamic algorithm
- **CHANGE -:** with this button, it is printed the representation of the previous configuration of the dynamic algorithm. If this button is pressed a lot of times, it will be printed the first configuration of the dynamic algorithm, which is the static configuration one.

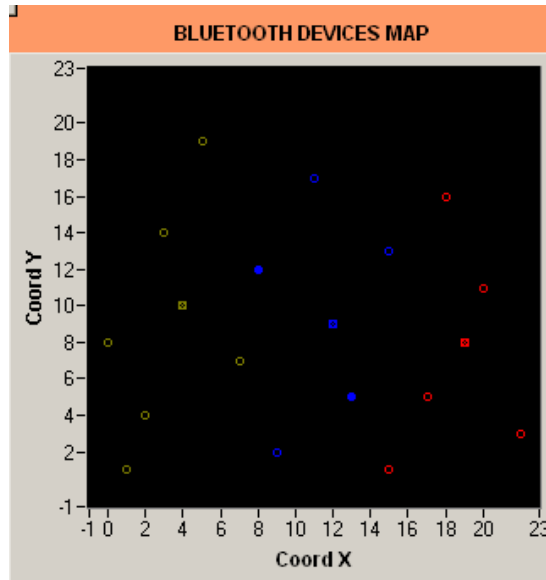


Figure 34: Configuration Bluetooth Devives

4. Piconets Members Part: this is a continuation of part 3, where the user can see how the configuration is: it can be seen, for each piconet of the scatternet, which device is the master and which ones are the slaves. It can also be seen the bridges devices between two piconets, because these devices appear in both piconets.

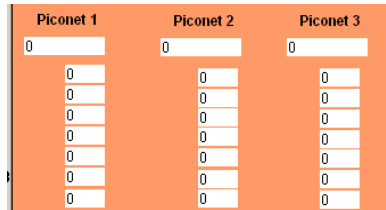
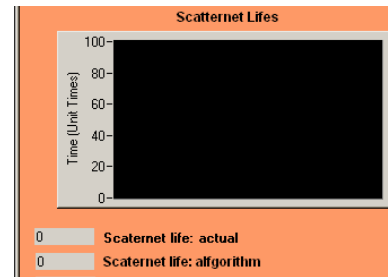


Figure 35: Piconet members

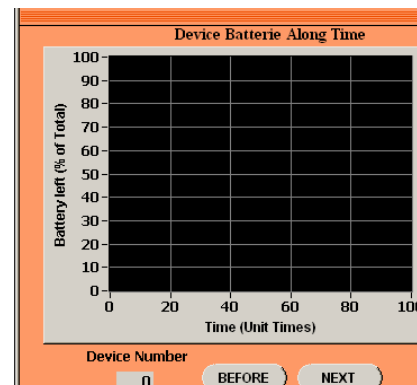
Whenever, the buttons of part 3 are pressed (ACTUAL, ALGORITHM, CHANGE +, CHANGE -), the indicators will change to show the configuration of the selected option.

5. Scatternet Lives Indicator: here, it will be compared the life (duration) of each algorithm: random, static and dynamic. There are two ways to do the comparison:

- Looking to the chart with vertical bars, where each bar represents the life of each configuration
- Reading the Number Boxes. There is a Number box for each configuration.



6. Device Battery Consumption: this is a chart where it can be seen the battery



consumption along time of the devices. The battery value is expressed in % of the maximum value, and it is considered that at the beginning all the devices have the maximum battery left.

It can be seen the chart of every device: just using the NEXT and PREVIOUS buttons, it will be seen the chart of the next or previous device. To know the device chart seen at the moment, there is a Number Box that indicates the number of the device represented at the moment.

7. Exit: when the EXIT button is pressed, the program is finished.

Figure 37: Battery along time

8. Caption: this caption (legend) is used to explain the symbols of the Bluetooth devices map (part 3):

- A master is represented by a square
- A slave is represented by an empty circle
- A bridge-slave is represented by a full circle

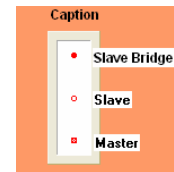


Figure 38: Caption

4.3 Example of use

To finish this part, it will be shown an example of how the program works. Our example consists of 19 devices. The steps of the example will follow like this:

1. **Data Entry:** This part is done in the part 1 of the interface.
 - a. First it is written 19 in the *Number of Devices* box, and press *Enter*.
 - b. Second: for every device, it is entered its coordinate x and y, and press *Enter*. When the coordinates of the last device are entered, in our case the device number 19, the algorithm will begin running. In a few seconds the results will be obtained.

Once, the number of devices and the coordinates are entered, this information can be checked in two ways:

- In the section 2 of the interface, *Bluetooth Devices Coordinates Part*, it can be seen the coordinates of each device.
 - In the section 3 of the interface, *Bluetooth Device Map*, it can be seen a two-dimension representation of the devices (printed in white), where the devices are free and there is not piconet/scatternet made yet. This initial view will change as soon as one of the buttons is pressed (ACTUAL, ALGORITHM, CHANGE + or CHANGE -). The devices will be part of a piconet/scatternet then.
2. **See the results:** as soon as the number of devices and their coordinates are entered, the algorithm will begin running. The running time will fluctuate between a few seconds and a couple of minutes, depending on the number of

devices used, and of course on the speed of the computer used. The results to analyze are:

- **The different piconet/scatternet configurations:** pressing the buttons ACTUAL, ALGORITHM, CHANGE + and CHANGE -, it will be seen the configurations created for each case, in the *Bluetooth devices map* (spatial view), and in the *Piconets Members part* (members of each device).

It is easy to see how the random configuration is not a logic one: there are devices connected to a master which is farther than another one, the piconets are not connected by the best bridge... However, the configurations created with the static and dynamic algorithms look more logical: the slaves are connected to the closest master, the masters are in strategic positions to create the scatternet, the piconets are connected by the easiest way...

- **The life of each configuration:** it can be checked in the *Scatternet Life Indicator* the life of each configuration. This can be done by reading the vertical bars of the graph, or just reading the Number Boxes for each algorithm (current, static and dynamic). It can be appreciated in the figure that the life of the static algorithm is quite higher than the random, and the dynamic is much higher than the static.
 - **Battery Evolution of each device:** in the *Device Battery Consumption* graph, it can be seen the consumption of each device along time (to change and see the chart of another device, it is just used the NEXT and PREVIOUS buttons). In the graph there are 3 lines, one for each algorithm (blue for the current algorithm, green for the static and red for the dynamic). Each configuration finishes when one device is with no battery left. In our example:
 - Current algorithm finishes because the device number 3, which has been working as a master, spends all its battery.
 - The static algorithm finishes because the device number 7, that has also been working as a master, spends all its battery
 - The dynamic algorithm finishes because the device number 11 has spent all its battery.
-

BIBLIOGRAPHY

Libros:

- Bluetooth, connect without cables, Jennifer Bray and Charles F. Sturman, PH PTR.
- Sistemas de Instrumentación. Dep. Eca. Univ. Alcalá
- Sistemas de Adquisición de Datos. Dep. Eca. Univ. Alcalá
- Specification of the Bluetooth System. Bluetooth™

Hojas Características:

- Bluecore 2 External
- Cambridge Silicon Radio CSR Bluelab development kit

Artículos IEEE:

- Distributed topology Construction of Bluetooth Personal Area Networks. Theodoros Salonidis, Pravin Bhagwat, Leandros Tassiulas, and Richard LaMaire
 - Local Positioning for Wireless Sensors Based on Bluetooth. Javier García Castano, Michael Svensson, Mikael Ekstrom
 - An adaptive Power-Conserving service discipline for Bluetooth. Hao Zhu, Guobong Cao, George Kesidis and Chita Das
 - Energy Efficient Bridge Management Policies for Inter-Piconet Communication in Bluetooth Scatternets. Ranganath Duggirala, Roy L. Ashok and Dharma P. Agrawal
 - Power- Efficient and QoS-Aware Scheduling in Bluetooth Scatternet for Wireless PANs
-

