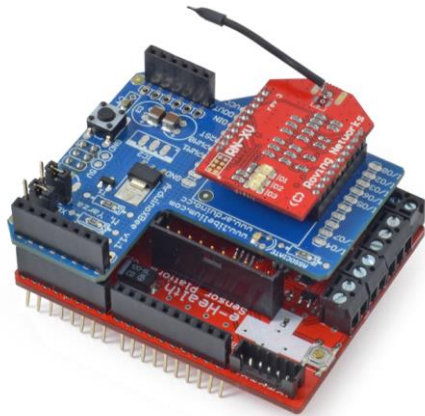




UNIVERSITAT DE
BARCELONA

Wireless Biodevices and Systems

Wireless Biodevices Based on Arduino



A. Garcia Miquel

A. Salas Barenys

N. Vidal Martínez

J. M. López Villegas

1. INTRODUCTION

In the next two sessions we will study the elements that form a wireless biodevice (WB) from a global technical perspective. The goal of this laboratory work is to implement a WB based on an eHealth platform for Arduino.

The aspects considered in this laboratory work are:

- the mounting and assembly of the different elements of a WB;
- introducing the programming of Arduino boards with the Processing language;
- the wireless transmission of biological data via different types of communication protocols (Wi-Fi, Xbee and Bluetooth).

At the end of the laboratory sessions you will be familiar with the components that make up a WB, both hardware and software, and the different stages of the design.

1.1. MATERIAL

- Arduino UNO board
- Support boards (signal conditioning and communication)
- Transducers or sensors (temperature, pulse oximeter, electrodes, etc.)
- Communication modules: Wi-Fi, Bluetooth, Xbee
- AB-type USB cable
- PC
- Smartphone or tablet with the Android operating system

1.2. PLATFORM DESCRIPTION

The **eHealth Sensor Platform V2.0 for Arduino and Raspberry Pi** developed by **Libelium**[®] has been chosen for these laboratory sessions. Click on the link below for further details:

https://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical#step6_3

The versatility of the platform permits us to connect up to 10 types of biomedical transducers and set wireless communication through 6 different technologies (Figure 1).



Figure 1. eHealth Sensor Platform V2.0 for Arduino (Libelium®)

The platform includes the following elements (Figure 2):

- Transducers (or sensors): The elements that convert a physical magnitude into an electrical one.
- Support boards
 - Signal conditioning board: Contains all the electronic components needed to read the signals from the transducers.
 - Wireless communication board: Allows the assembling of wireless communication boards on the Arduino board.
- Microcontroller (MCU): An Integrated Circuit capable of executing the instructions stored in its memory. It contains a central process unit, a memory and in/out peripherals.
- Power management
- Serial console: Allows communication with the computer.
- Communication modules: Allow wireless communication through Wi-Fi, Xbee or Bluetooth protocols.

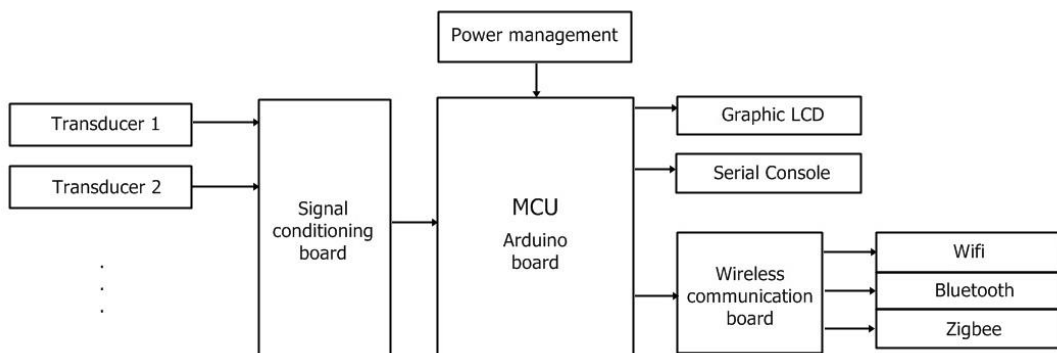


Figure 2. WB elements

1.3. USING ARDUINO

Arduino is an open code electronic platform specially designed to facilitate its use in multidisciplinary environments. The hardware consists of a board with an integrated MCU of the ATmega family, characterized by its simplicity and low cost. The software is based on an integrated development environment (IDE) that uses the Processing programming language.

The Arduino board model used in these laboratory sessions is the Arduino Uno R3 (Figure 3). The features it offers, such as the number of digital in/out pins, the communication ports, the analogical inputs or the memory, vary depending on the model.

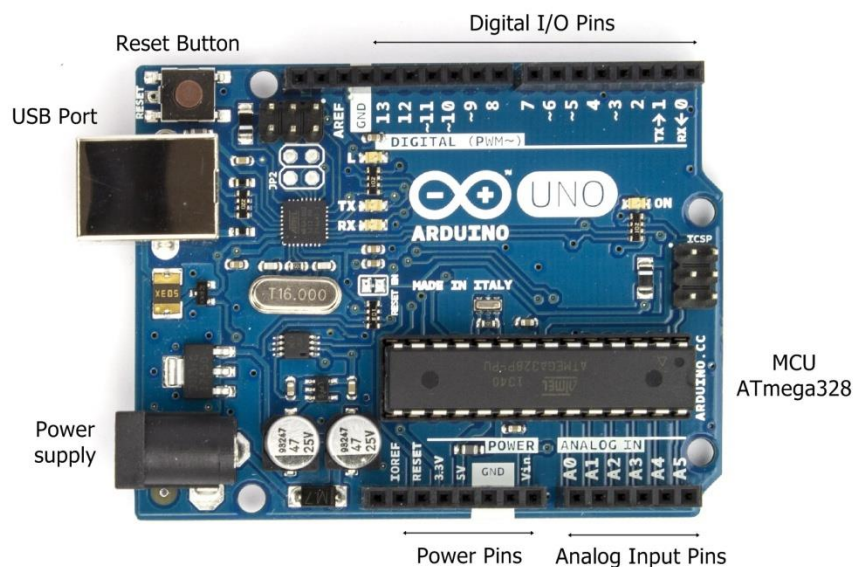


Figure 3. Arduino UNO R3 (Libelium®)

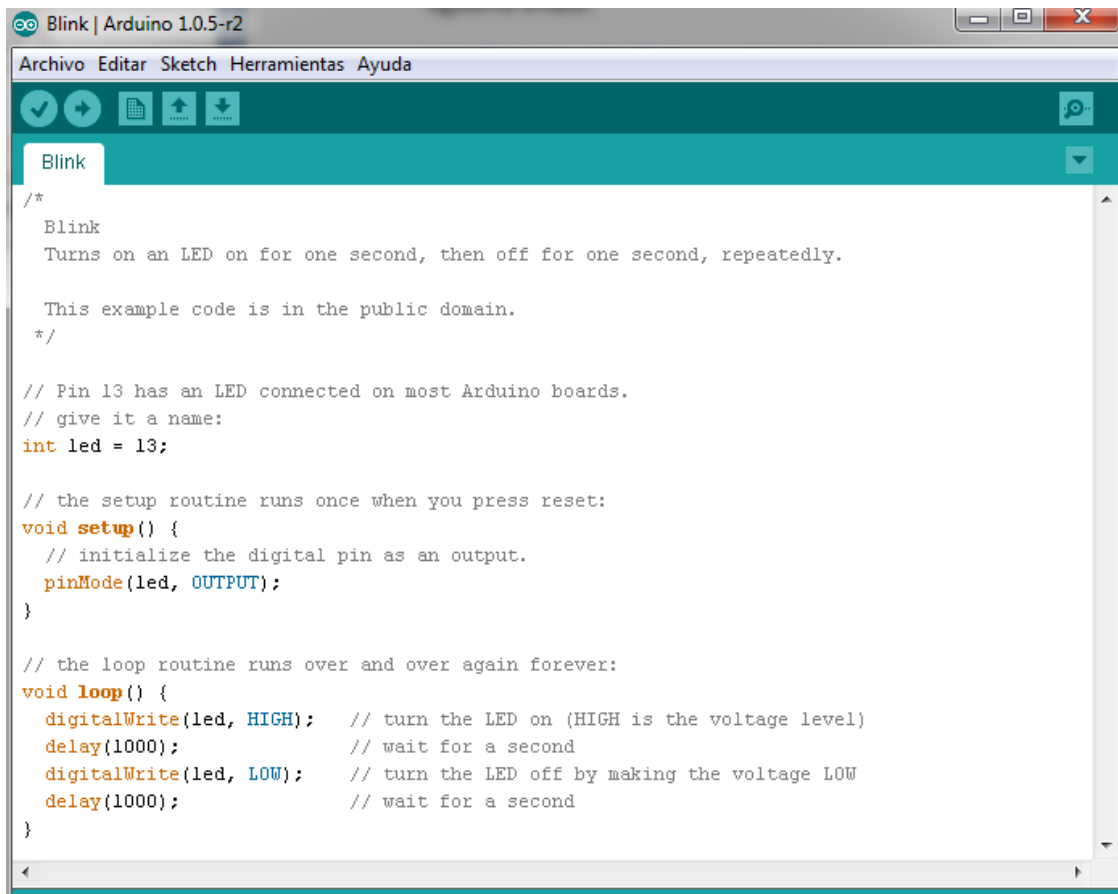
All the required information about this device can be found on the Arduino website (www.arduino.cc).

To install the development environment used in Arduino, we just need to download the latest version from the link:

<https://www.arduino.cc/en/Main/Software>

If you need help with the functions used in the Processing programming language, you can visit this link:

<http://arduino.cc/en/Reference/HomePage>

The image shows a screenshot of the Arduino IDE interface. The window title is "Blink | Arduino 1.0.5-r2". The menu bar includes "Archivo", "Editar", "Sketch", "Herramientas", and "Ayuda". Below the menu bar is a toolbar with icons for file operations and execution. The main text area contains the following code:

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
  */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);             // wait for a second
  digitalWrite(led, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);             // wait for a second
}
```

Figure 4. Sample code in the Arduino IDE.

A sample of the Arduino IDE and the Processing language is shown in the example in Figure 4 . Can you figure out what this code does? What are void setup() and void loop() and what are they used for?

The following link is to a video of one of the creators of Arduino, Massimo Banzi, where some of the applications developed on this platform are listed:

http://www.ted.com/talks/massimo_banzi_how_arduino_is_open_sourcing_imagination

As you can see, you can do a lot of things with Arduino in a simple, low-cost and fun way. In addition, hundreds of handmade projects are available on the Internet; you can download some code, buy things and let your imagination fly!

1.4. USING THE LIBRARIES

In order to connect the transducers to the Arduino board, a support board is needed (Figure 5). It has all the electronic components needed to gather the signals received from the transducers and send them to the MCU.

However, we need code that allows us to use the transducers with the support board: the library. In essence, a library is a lot of lines of code that are already programmed and are organized with functions. These functions allow the developer not to worry about programming parts of the code. In our case, the libraries mean we do not need to know how to program the device to read the values of the data from the transducers.

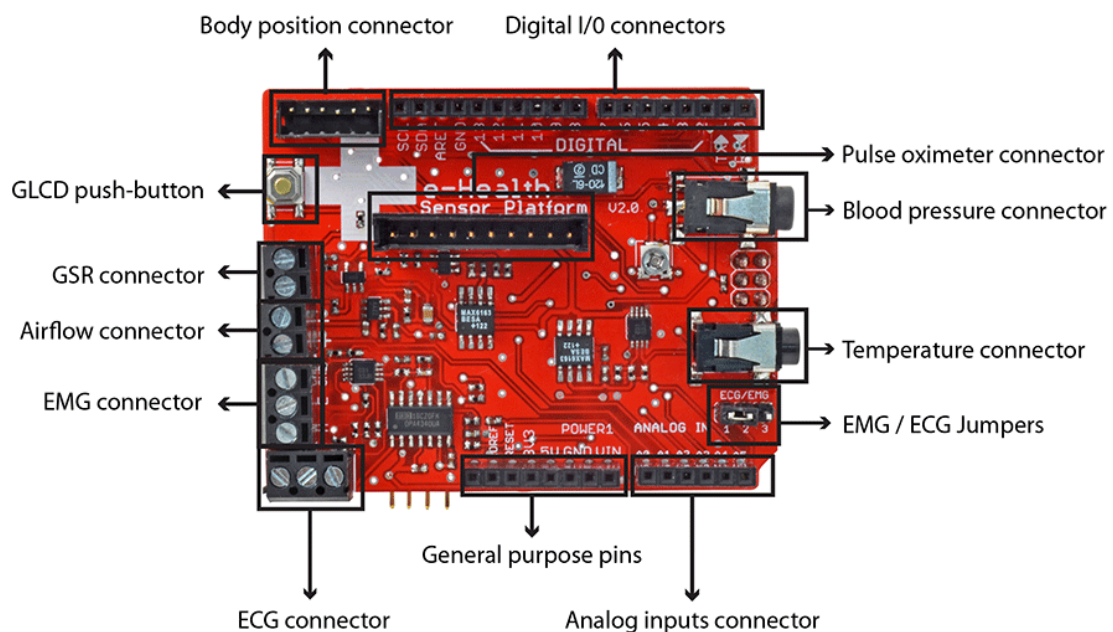


Figure 5. Signal conditioning board (Libelium®)

1.5. SENDING THE DATA

There are different ways of transmitting data from our device to another device, and there are many display options. We will see some of the most common in these laboratory sessions.

1.5.1. Serial Port: PC

Connection with the computer is via the same USB cable that we use to power the Arduino board. We will be able to visualize the data on our PC.

1.5.2. Wireless communication

A support communication board is necessary for wireless data transmission (Figure 6). The desired wireless protocol module will be mounted on it.

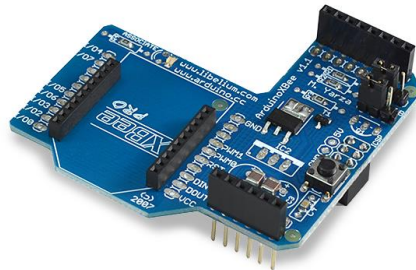


Figure 6. Wireless communication board (Libelium®)

1.5.2.1. Wi-Fi

The Wi-Fi Roving RN-171 module (Figure 7 (a)) will provide wireless connection of our device with any other device equipped with Wi-Fi, such as a mobile phone. In addition, we can connect our device to a Wi-Fi network and send the data to the cloud.

1.5.2.2. Bluetooth

With the Bluetooth module (Figure 7 (b)) we can send the data to a smartphone or a PC.

1.5.2.3. Xbee

The Xbee module (Figure 7 (c)) will allow us to send data wirelessly from one Arduino to another.

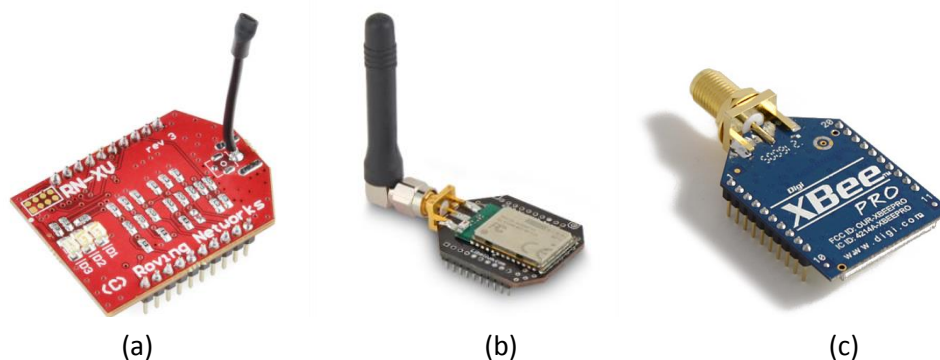


Figure 7. Wireless communication modules: (a) Wi-Fi, (b) Bluetooth and (c) Xbee

2. EXPERIMENTAL PART

2.1. INSTALLATION

First, the Arduino IDE must be installed on the PC, following the instructions at:

<http://arduino.cc/en/Guide/HomePage#UyhwAfl5Pzl>

Once the IDE is installed, the Arduino board must be connected in the computer via the USB cable. Open the program and ensure that the proper board (Arduino UNO) and port are chosen in the *Tools* menu.

The second step is to install the eHealth libraries developed by the company Libelium®, which contain the code necessary to translate the signal values of the sensors. You have to download version 2.2 of the libraries for Arduino and then copy them into the right folder. Please, follow the instructions here:

https://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical#step3_1

Now you can open the Arduino IDE and check that you have the programming codes in the eHealth example folder.

2.2. ASSEMBLING

The next step consists of assembling the different parts. A detailed description of the process can be found on the website of the device:

<https://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical/>

First, you have to mount the eHealth support board (the red board) above the Arduino board. Then, you can check on the website where to connect the transducers.

2.3. SENDING THE DATA: LOCAL COMMUNICATION

Now you can spend a few minutes checking one or two sensors by using the correspondent code examples that you can find in the eHealth example folder of the Arduino IDE.

2.3.1. Serial Monitor

Example: Temperature sensor

In the eHealth example folder, there is a program (or sketch) called **TemperatureExample**. The sketch must be opened and built on the board through the arrow in the upper part of the screen.



Unless errors occurred while building, the *MonitorSerial* can be opened from the upper right side of the screen.



A blank screen appears and the temperature values are displayed.

If errors happened, check the number in the right lower corner of the *MonitorSerial*. This number is the speed of the transmission between the Arduino board and the computer, and has to be the same as the number at the beginning of the code.

If the error persists, a variable might need to be included in the eHealth library. To solve this, go to the *eHealthDisplay.h* file (in the library folder) and copy the following code at the top:

```
#ifndef prog_uint8_t
#define prog_uint8_t const uint8_t
#endif
```

Other errors consist of wrong connection of the transducers, incorrect assembling of the boards and issues with the communication ports, among others.

If everything is ok and you visualize the data, you can now check the correct functioning of the temperature sensor by rubbing it. Do not worry if the sensor tells you that you are at 42 degrees; sometimes the sensor is not well calibrated and the values are not accurate. If you are interested in how to calibrate the sensor, you can find the process on the device website (see link above).

Once you have completed the visualization of at least one transducer in the *MonitorSerial*, you can continue with the next step of the session, where a more sophisticated visualization of the data will be achieved.

2.3.2. Real-time plot

For better visualization of the data on the computer, we can use complementary software, such as the **Realterm** and **KST** software packages. The former generates a .txt file with all the numerical data received through the communication port where the Arduino is connected. The file is read by the second program which makes a plot in real time. There are other software packages that can be used for the same purpose.

The procedure for the real-time plot visualization is explained below.

Example: ECGExampleKST

In this example, the **ECGExampleKST** sketch will be used, with the Arduino board connected to port COM 3. You can do the same with the EMG sensor (sketch **EMGExampleKST**) and conductivity (sketch **GSREExampleKST**).

The transmission rate is expected to be 9600 bauds in the serial monitor of the Arduino as well as in the sketch code lines, as indicated in Figure 8.

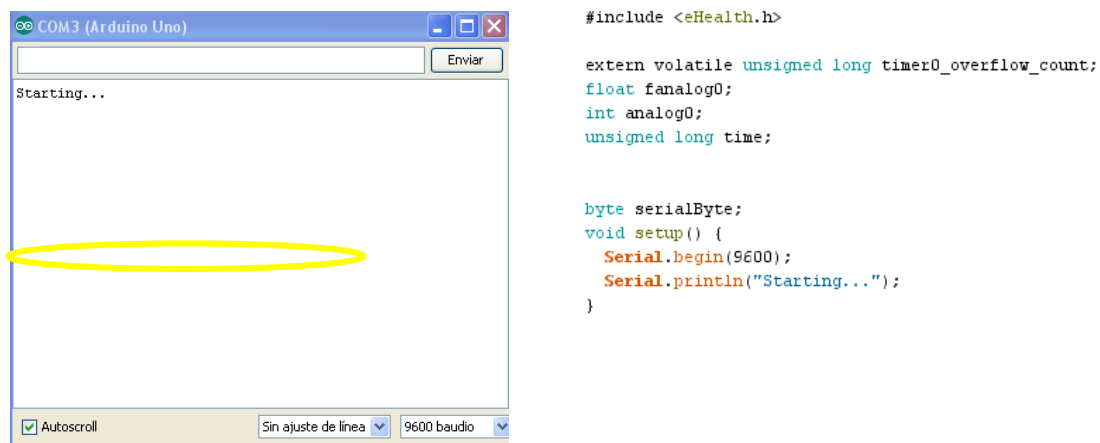


Figure 8. The 9600 bauds are observed both in the *SerialMonitor* and in the sketch

The first step is to

download the Realterm software from this URL:

<http://sourceforge.net/projects/realterm/files/>

Then, you have to follow these steps in order to save the data in a file:

In the Port tab, click the button Clear. Then, select the corresponding Baud value (9600 in this case), the corresponding port, and finally, *Open* (Figure 9).

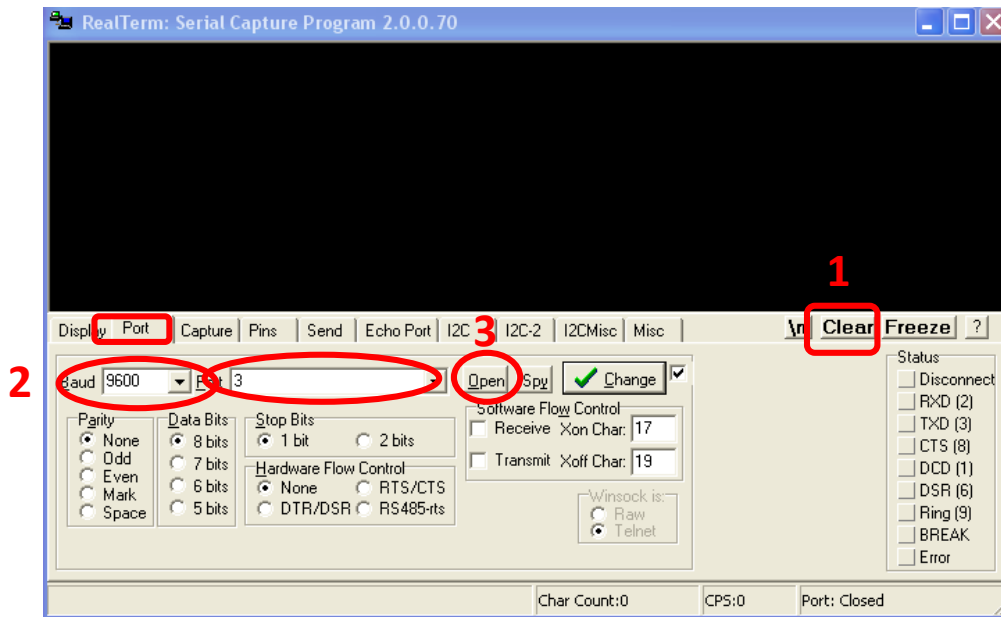


Figure 9. Initial screen of RealTerm software

Now, you should see “Starting...” in the black screen (Figure 10).

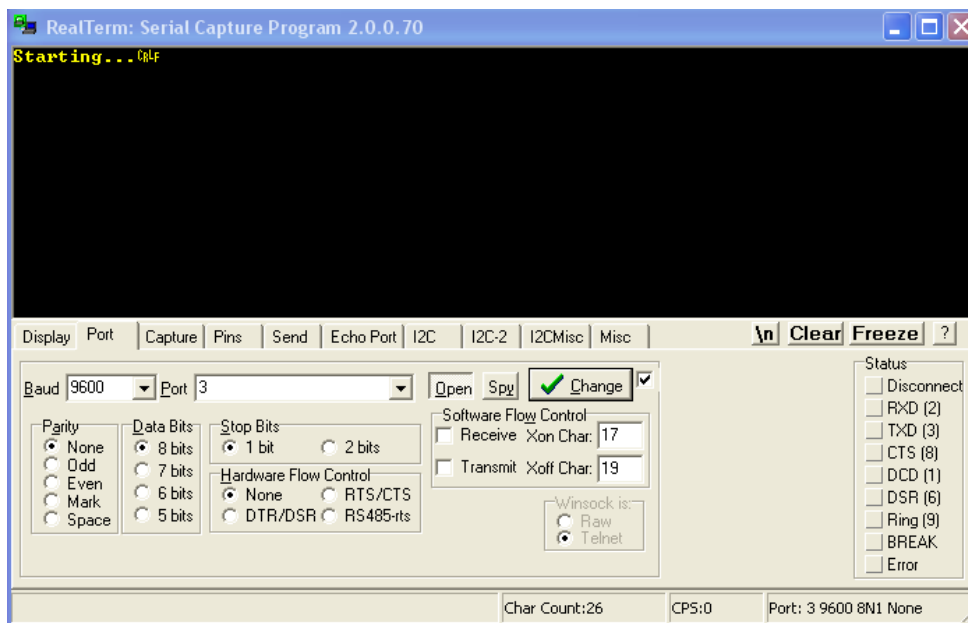


Figure 10. Visualization of the beginning of data capture

Next, on the *Send* tab, you must indicate *C* as the character to send (Figure 11).

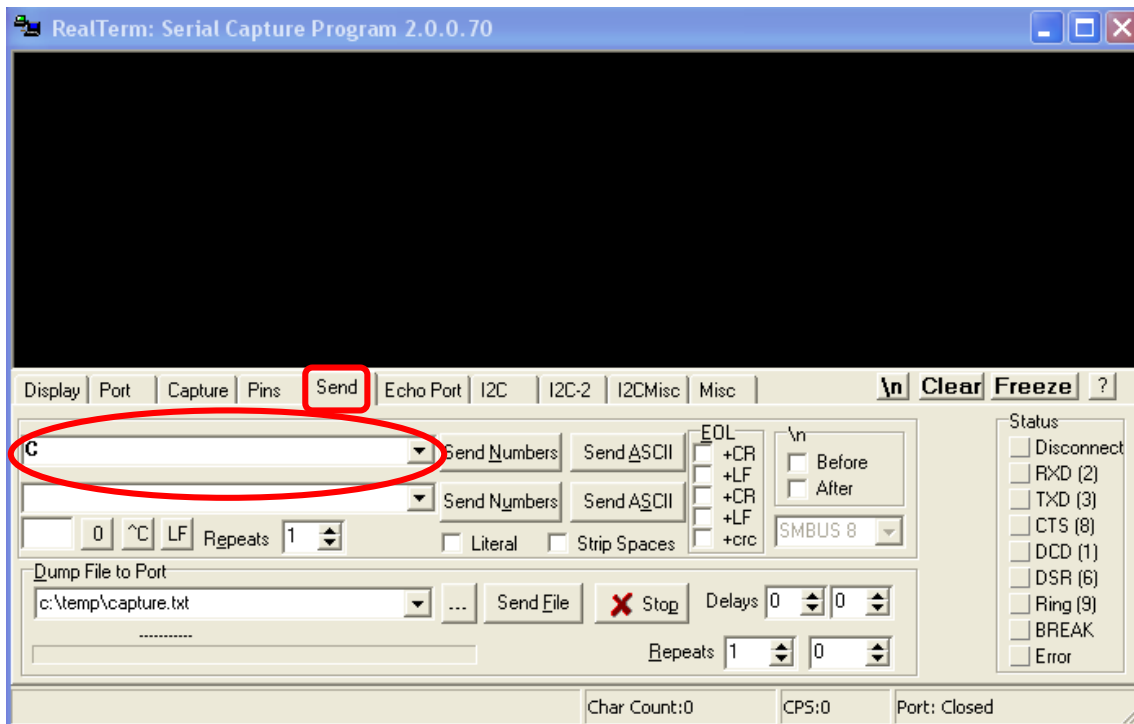


Figure 11. Sending the C character

The next step consists of clicking the *Send ASCII* button. Numerical values should appear in the upper screen (Figure 12).

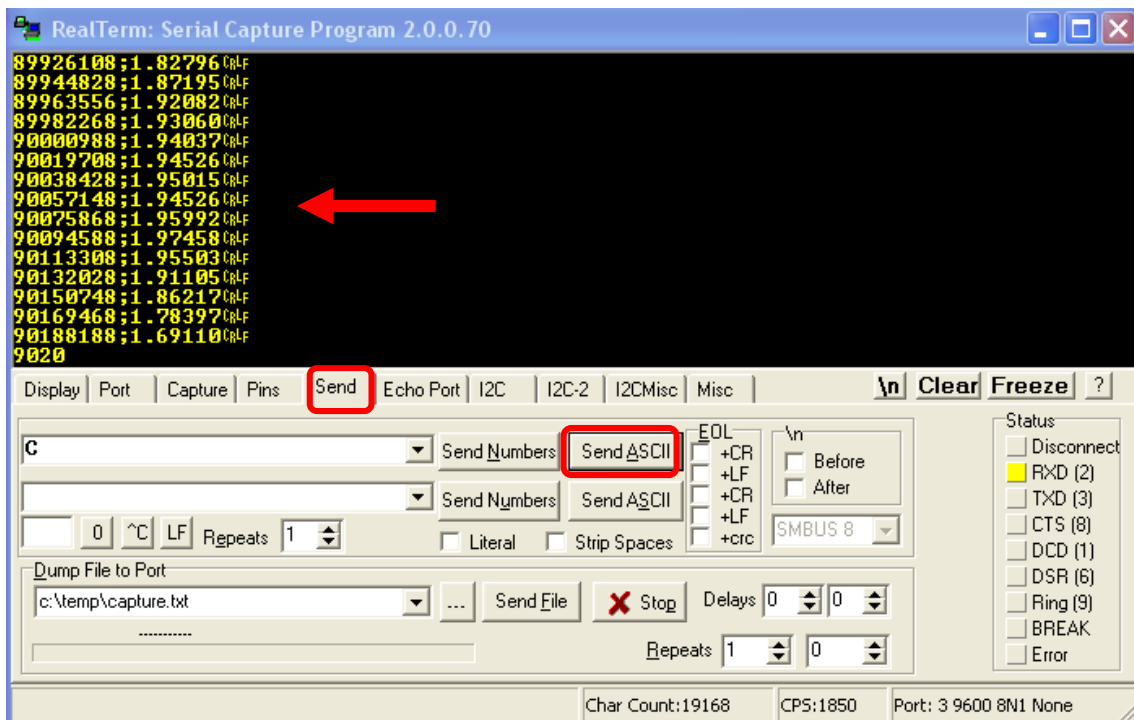


Figure 12. Reception of data

The name and location of the file to generate must now be specified (Figure 13). This can be done in the *Capture* label. In this example, the name of the file generated will be capture.txt and its location *G:\Documents and Settings\Mis documentos\Arduino*. Hence, it must be introduced as:

G:\Documents and Settings\Mis documentos\Arduino\capture.txt

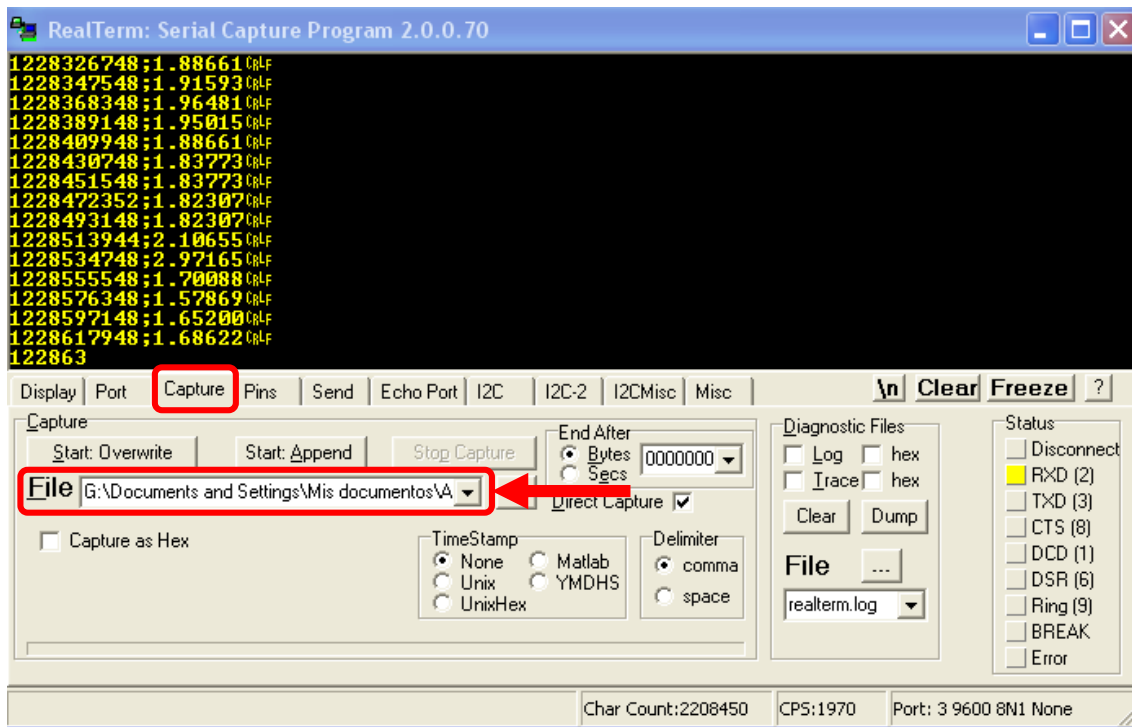


Figure 13. Selection of the destination file

In order to generate the file, click the button *Start: Overwrite* (to start) and *Stop Capture* (to stop). For a real-time display, just click *Start: Overwrite*.

Once the file is generated, KST will be used to show the graph. The software can be downloaded from the URL:

<http://sourceforge.net/projects/kst/files/>

Once installed, the first step is to access to the *Data Wizard* in order to select the data (Figure 28).



The file generated has to be introduced into the screen that appears. Then, go to *Configure...* (Figure 14).

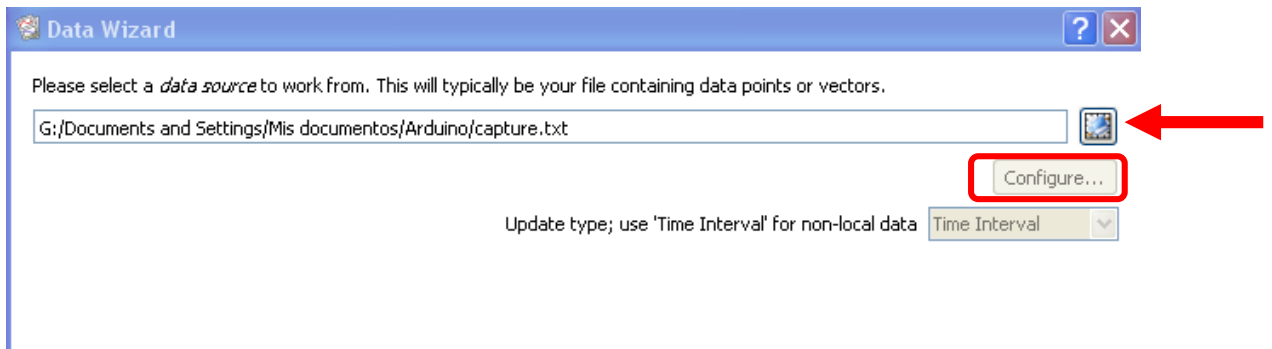


Figure 14. Data file access

In the configuration, the custom delimiter must be specified. In this case, to obtain both data series required for the graph, and taking into account how the values are defined by the sketch, “;” will be defined as the separator (Figure 15).

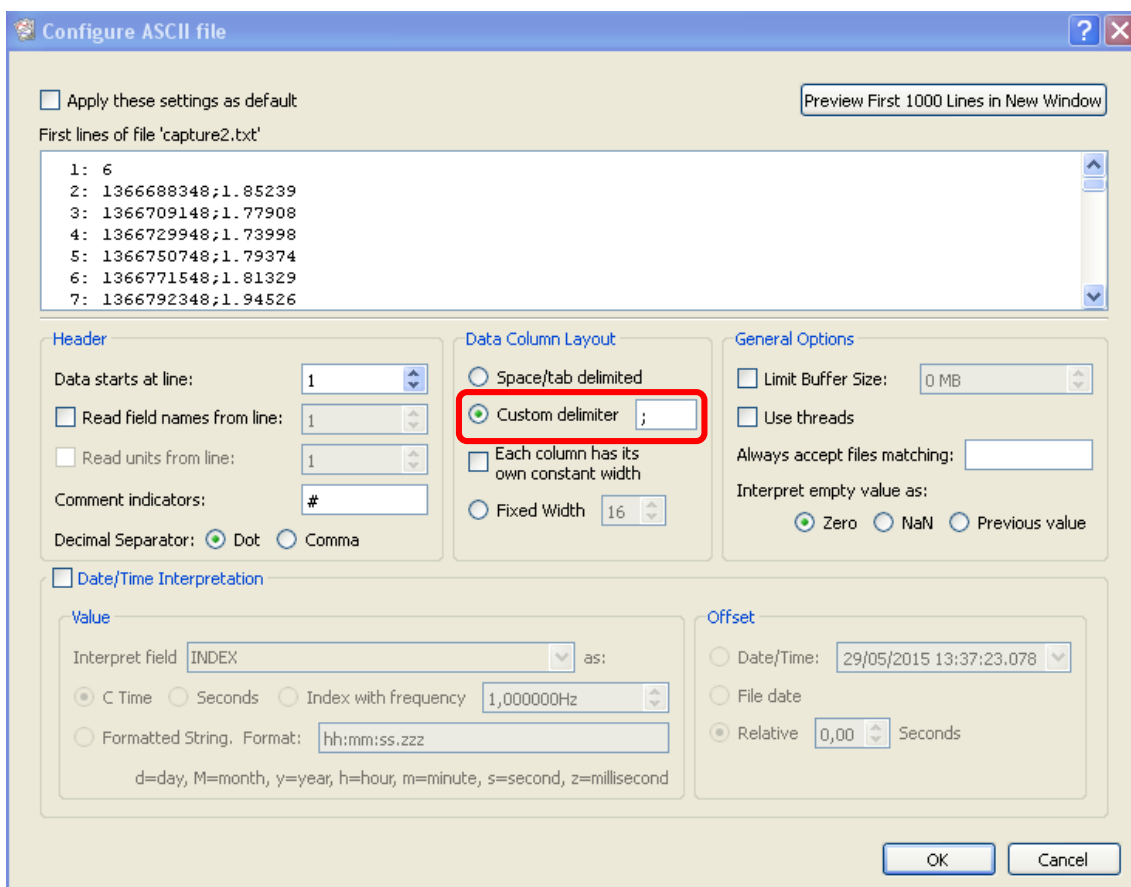


Figure 15. Delimiter selection

Then, the data stored in *Column 2* must be selected (Figure 16) as the data to display on the Y-axis. The data for the X-axis must be the data available in *Column 1* (Figure 17).

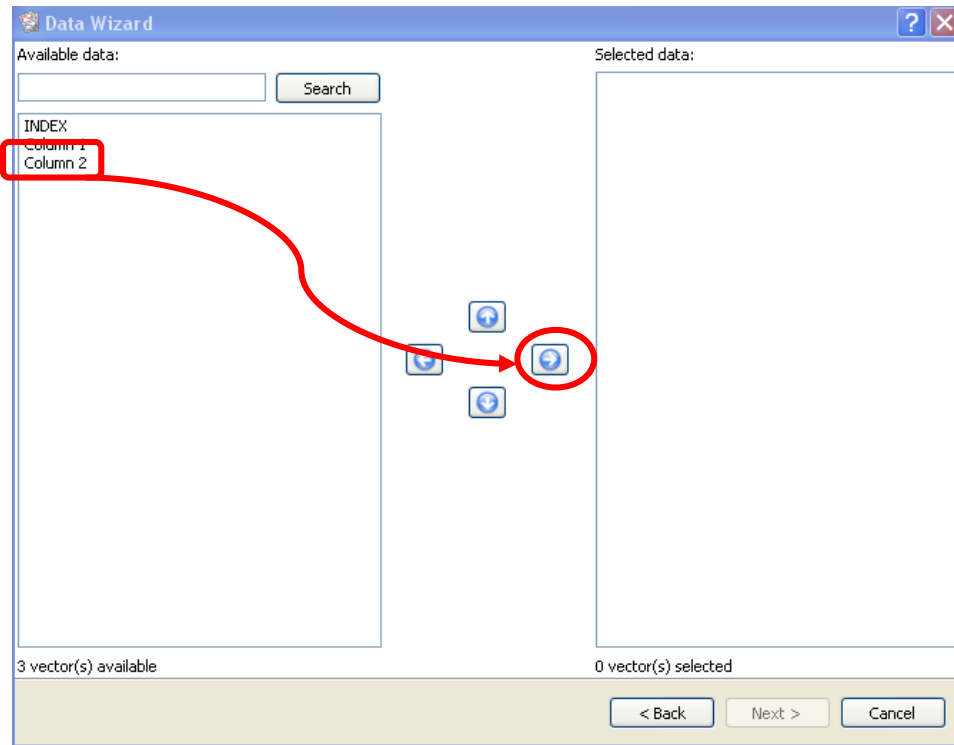


Figure 16. Selection of the column to display

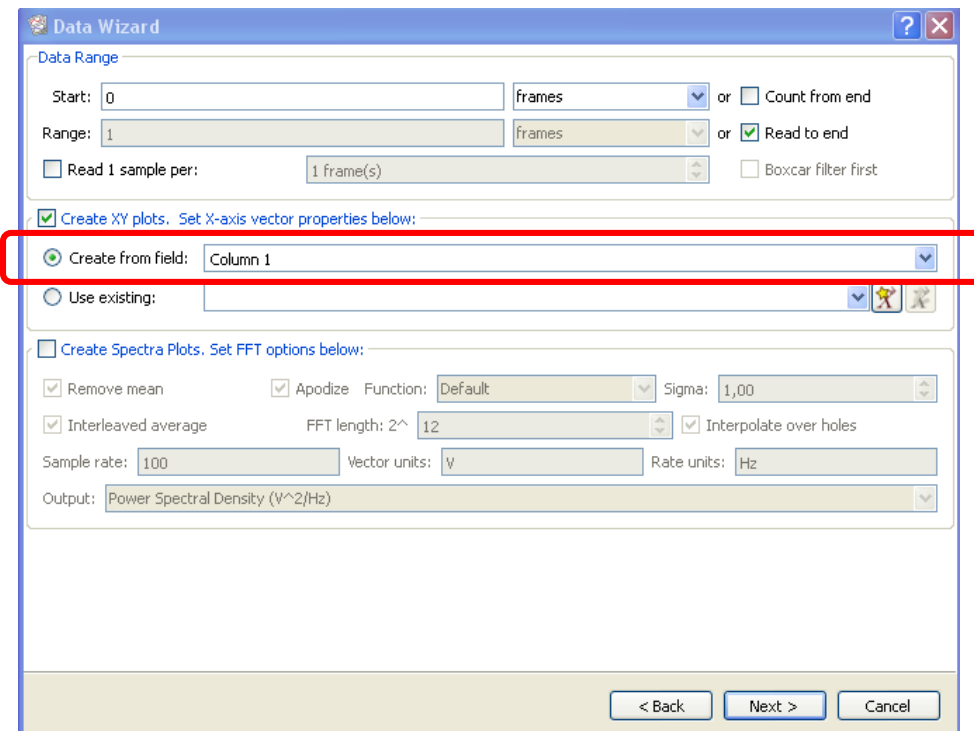


Figure 17. Selection of the data for the X-axis

Finally, when clicking *Finish* in the last screen, it something similar to the plot in Figure 18 should appear.

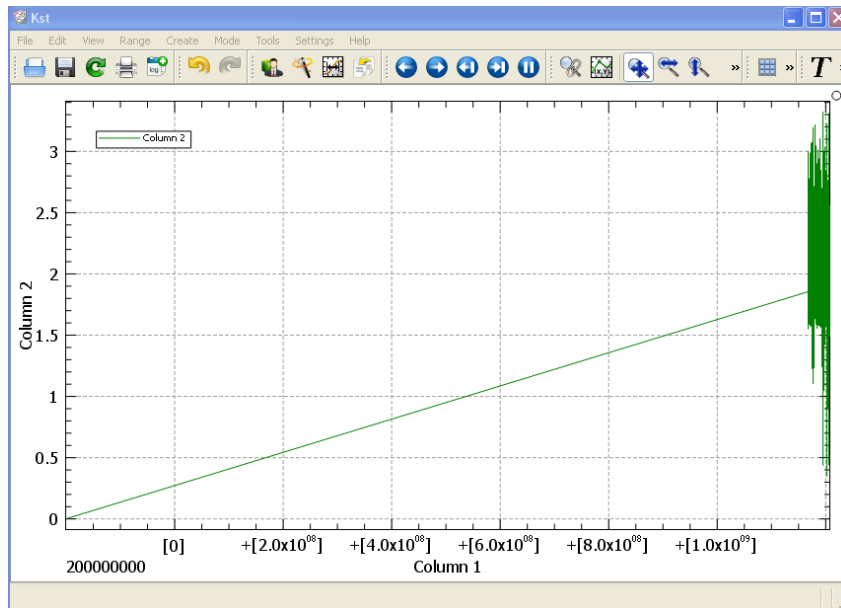


Figure 18, Initial plot

Then, you just have to adjust the zoom with the cursor in order to visualize the data obtained properly, as in Figure 19.

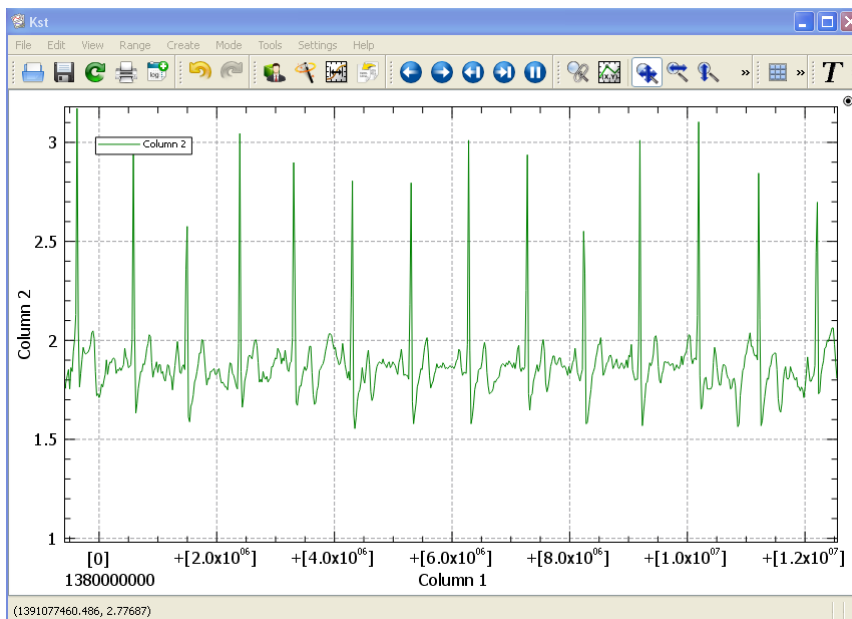


Figure 19. Adjusted plot

2.4. SENDING THE DATA: WIRELESS COMMUNICATION

In this part of the laboratory session we will send the data to another device using wireless technology: Wi-Fi, Bluetooth or Xbee.

2.4.1. Wi-Fi

Using the Wi-Fi communication protocol, it is possible to transfer the acquired data to other devices. In this case, we will send the data in two different ways:

- To a smartphone.
- To a web server.

2.4.1.1. Mobile App

Wi-Fi technology allows visualization of the biomedical data on a tablet or a mobile device, which is done using the sketch **AndroidAppPractica** uploaded from the virtual campus.

Moreover, the eHealth application developed by Libelium® must be downloaded from the link below and installed on an Android device.

<https://play.google.com/store/apps/details?id=com.libelium.ehealth&hl=es>

Figure 20 shows its appearance:



Figure 20. Some screens from the mobile application

Now, a Wi-Fi access point will be created from the mobile device (without an access key). The eHealth platform website might be helpful:

https://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical#step6_1

The name of the Wi-Fi network created must be specified in the Arduino sketch, in the following code line:

```
Serial.print("join WIFIANDROID\r"); check();
```

where WIFIANDROID is to be replaced by the name of the Wi-Fi network you have selected. Then, build the sketch in the Arduino board.

NOTE: Before uploading the code onto the Arduino board, take off the jumpers of the communication board (Figure 21). Afterwards, put back on the Xbee side jumper and restart the Arduino board by pushing the reset button (or disconnect and reconnect the USB cable).

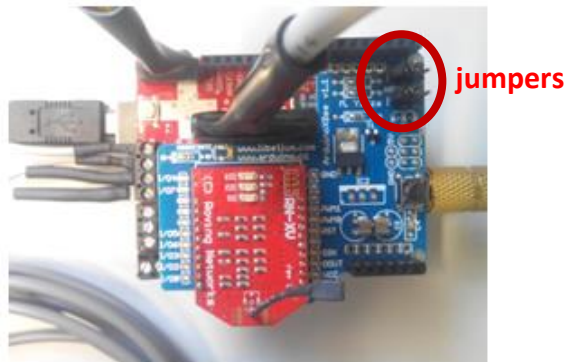


Figure 21. Location of the jumpers

Instructions for configuration of the Wi-Fi network and the data transmission to the mobile device can be seen on *MonitorSerial*. If the Wi-Fi module is connected in the proper network, its red light will turn off. If the connections have been successfully set, a green circle (instead of red) will be seen on the upper right side of the mobile application.

Some sensor values may not be correct since it is a test application that is not completely developed. For the same reason, errors may appear while charging the sketch in the following cases:

- Due to a bad connection between the boards.
- Due to an incorrect COM port selection where the Arduino is connected (disconnect and connect the USB cable).
- Because the Arduino serial monitor is open.
- No error has been identified while uploading the sketch, but it does not work. This may be due to the jumper position.

If an error appears during the connection between the smartphone and the Arduino, restart the Arduino board or reset the connection with the smartphone.

2.4.1.2. Web Server

Another application for sending data via Wi-Fi is the visualization of the data acquired by the Arduino in a web server. This option would be very useful in the case of, for example, a patient who is required to be remotely monitored from home. It must be taken into account that this is not a valid option for patients in critical conditions, since neither the robustness nor the security protocols are appropriate for the circumstances. Nevertheless, it is a good choice to acquire signals periodically from subjects in rural and remote areas, as well as from their homes, so that they do not have to move to the health center frequently.

A Wi-Fi network is necessary to connect the smartphone. It can be generated either from the smartphone, as in the last section, or by a router. The only difference between the cases is the implementation of the network parameters in the sketch.

Another point to take into account is the IP address which the platform must connect to. In this case, the connection must be to a server of the *Universtat de Barcelona* called *docenciatest* that is found in the IP address 161.116.95.246. These parameters are specified in the following code lines:

```
const char server[] = "161.116.95.247"; //IP Server
const char server_port[] = "80";
const char wifi_ssid[] = "WIFIANDROID";// WiFi network name
const char wifi_password[] = ""; //Do not put any password
```

where WIFIANDROID must be replaced by the name of the Wi-Fi network you have selected.

The sketch containing the code is in the campus and is called **WebServerPractica**. The sketch has to be uploaded, taking into account that you must take off the jumpers and put them back afterwards. Note that there is a variable called *user* which is related to patient identification, which means that more than one patient can be monitored simultaneously. This number must be changed depending on the corresponding laboratory group.

Figure 22 shows the information flow. The Arduino devices are connected to a Wi-Fi network and send the data to the web server, where an application stores it in a database. When the application is called by writing its address in the browser, the latest data are presented to the user (a doctor or a caregiver, for example):

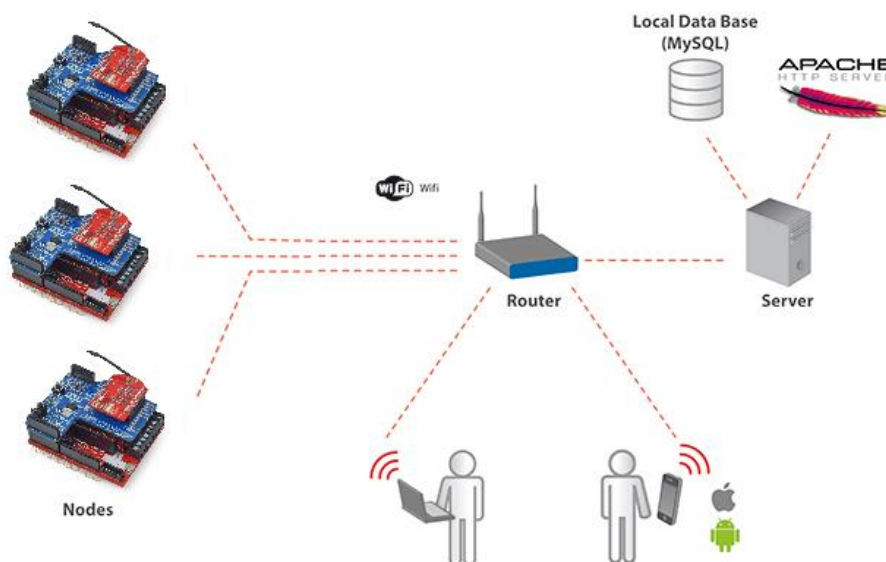


Figure 22. Elements involved in communication with a web server

The address of the application is: <http://161.116.95.247/~docent/ehealth-app/>

A screen like that shown in Figure 23 will appear.

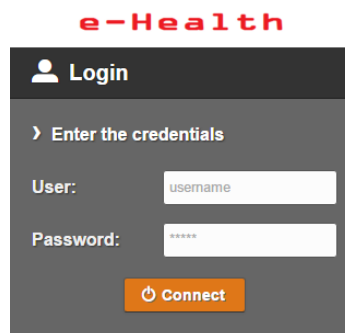


Figure 23. Initial screen of the eHealth application

The credentials to enter into the web application are:

User: admin

Password: 12345

Figure 24 shows the appearance of the web once the user is validated. The red light indicates that Patient 1 is not connected; a green light indicates that Patient 2 is online.

It takes a while to visualize the data, since this is only a web application in an early stage of development. In addition, data updating is not automatic: a step back must be taken and the data entered again in the sensor section to get the new values.

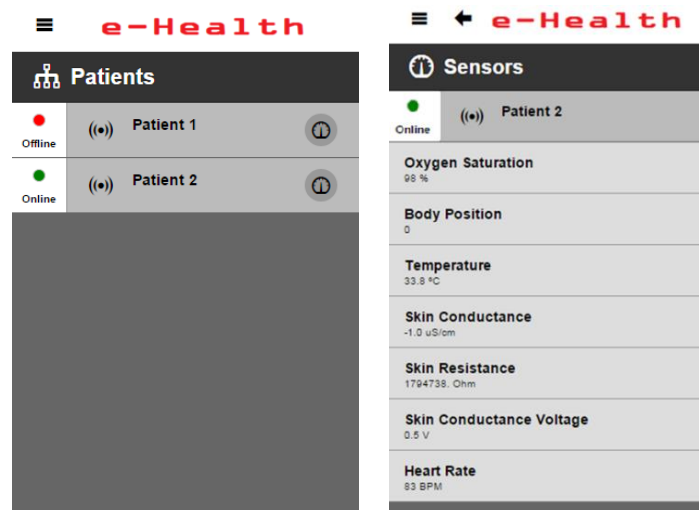


Figure 24. Visualization of the data acquired on the web application

2.4.2. Bluetooth

Nowadays, the majority of the mobile phones have the option of setting Bluetooth communication with another device. This technology will be used to link a smartphone with our device. The Arduino will acquire biomedical signals and send them using the Bluetooth protocol to the smartphone. There, the numerical data will be displayed through an application called *Bluetooth Terminal*.

For this software, the sketch **BluetoothModificadoPractica** will be necessary, as well as the application for Android Bluetooth Terminal. The sketch **BluetoothModificadoPractica** is in the campus, while the application can be downloaded from the Android device (smartphone or tablet) Play Store. The appearance of the application is shown in Figure 25.

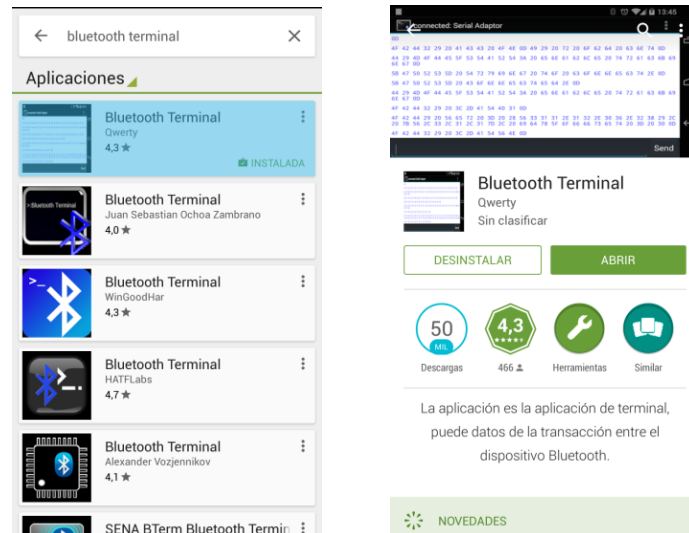


Figure 25. Search and download process of the Bluetooth Terminal application

At this point, the Arduino board has to be assembled to the other components and the Bluetooth module. Remember the position of the jumpers in the communication platform before uploading the sketch (Figure 21).

While charging the sketch, some compilation errors may appear. This is because a few errors have been included in the code and some code lines have been eliminated, to prevent correct functionality.

You have to modify the sketch **BluetoothModificadoPractica** in order to visualize the data from the sensors to the connected smartphone. The result should be similar to Figure 26.

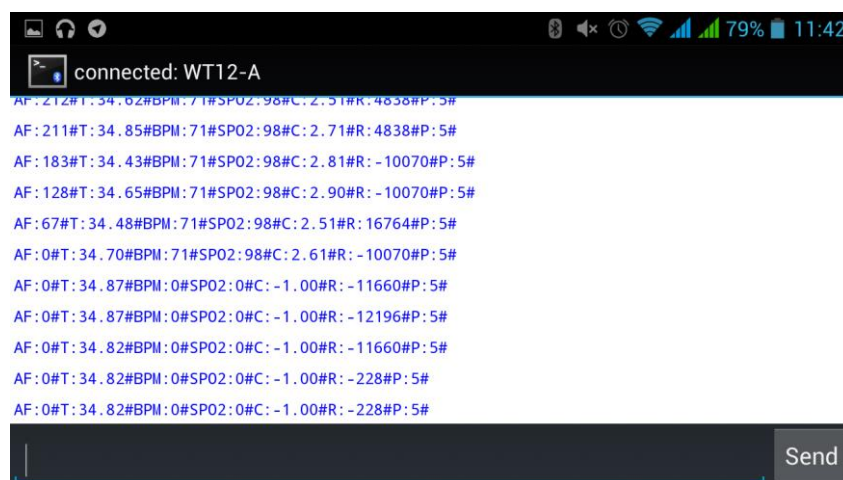


Figure 26. Visualization of the data in the mobile via Bluetooth communication

To make communication between the smartphone and our device possible, the Bluetooth function must be activated in the smartphone which should recognize the Arduino device.

The next step is to open the *Bluetooth Terminal* app and set the Bluetooth connection from there. To do this, you have to access the application menu and select *Connect a device-Insecure*. In this way, it is not necessary to introduce a password. If a password is requested, try "00000". The Bluetooth device is selected and shortly after the connection is set, the values appear in the smartphone.

NOTE: The *Bluetooth Terminal* application may lose connection while the smartphone is in motion. Thus, when data is being displayed, it is best to not move the phone. If this happens, reset the connection.

2.4.3. Xbee

Using the Xbee communication protocol it is possible to connect two Arduino devices to display on one of them the data acquired by the other. The emitter will acquire the biomedical signals and sends them through the XBee-PRO ZB module to the receiver device. That device will be connected to the computer where the data will be visualized using the serial monitor of the Arduino.

Related to the software, two sketches will be needed, one for each Arduino:

- **ReceiverZigBee:** Arduino receiver (in the campus).
- **ZigBeeCommunicationExample** Arduino emitter (in the eHealth examples).

The Arduino receiver just needs to be assembled with a communication board and the XBee-PRO ZB module. Optionally, the antenna can be added to increase the gain and hence the communication range. The sketch must be built with the jumpers removed from the communication board, then they must be replaced in the Xbee position (Figure 21).

The Arduino emitter must be mounted with the eHealth board, the sensors, the communication board and the XBee-PRO ZB module.

In order to display the data from the receiver, it must be connected to a PC and the *MonitorSerial* must be accessed, from where all the data acquired by the other terminal will be visualized.

2.5. BUILDING YOUR OWN SENSOR

Until now, we have worked with the eHealth board (the red board) and the libraries designed and provided by the company Libelium®. In this final part of the laboratory work, this support board will be left aside and we will create our own circuit for our own temperature sensor. To do this, a temperature sensor (NTC-K164 thermistor of 2.2 k Ω , similar to that in Figure 27) will be used.



Figure 27. NTC thermistor

The chosen circuit is the Wheatstone bridge (Figure 28). The main idea of the circuit is that when all the resistances have the same value (nominal value of R_x), the voltage of both tension dividers (nodes B and D) are equal. This happens when we are at room temperature, about 22°C. If the temperature changes, R_x changes, and then there is a voltage difference between nodes B and D. In our case, the voltage between nodes B and D will be determined by the Arduino analog inputs (such as A0 and A1).

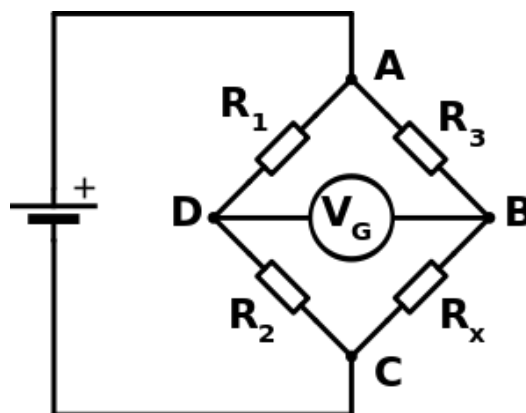


Figure 28. Wheatstone bridge

The additional material needed in this part is the following:

- Multimeter
- Protoboard
- Screwdriver
- Thermistor NTC
- 3 x Potentiometers 10 k Ω
- Resistor 10 k Ω
- LED
- Wires

In order to implement the circuit, three potentiometers (or variable resistances) of 10 k Ω (Figure 29) will be used for the three resistances of the Wheatstone bridge (R1, R2, R3). It will be necessary to adjust them at the nominal value of the NTC thermistor (2.2 k Ω). To do that, use the multimeter, reading between one of the side pins and that in the middle. With the screwdriver, modify the potentiometer until it has a 2.2 k Ω resistance. Once the three potentiometers are properly configured, they should be connected, together with the NTC thermistor, using the protoboard (Figure 30), and following the structure of Figure 28.

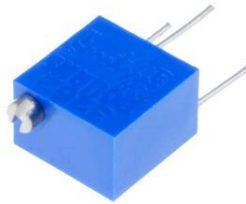


Figure 29. 10 k Ω potentiometer

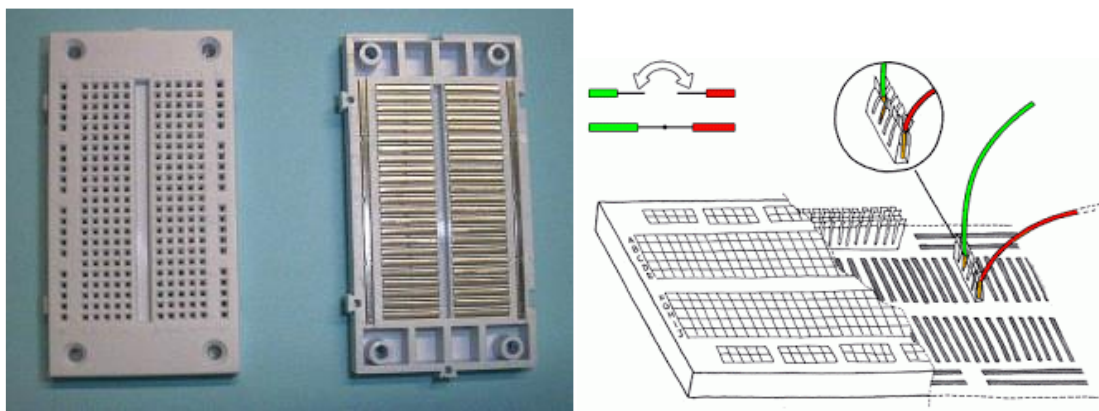


Figure 30. View of the protoboard internal connections.

The power of the circuit will be provided by the Arduino through the 5 V and GND outputs.

You have to make a program (sketch) with the following orders:

- 1) Read the analog ports corresponding to nodes B and D (A0 and A1, for example).
- 2) Calculate their difference and scale this value according to the sensor characteristic. The code line to scale the temperature is something like (although it can depend on several factors):

```
temperature = t0 + 0.214 * diff;
```

where *diff* is the difference between the analog inputs and *t0* is the room temperature (usually around 22°C).

- 3) Then, send the temperature value to the PC through the serial port.

Warm/cool the sensor to check its behavior.

NOTE: Look at functions `pinMode()` and `analogRead()` in the Arduino Reference.

Once you have checked that the temperature is being properly read through the serial port, and if you still have time, some exercises are proposed:

1. Connect an LED with a 10 kΩ resistor to one of the digital outputs of the Arduino, so that when the temperature surpasses a certain value (38 °C, for example), the LED turns ON as an alarm. When the temperature goes back to below 38 °C, the LED should turn OFF again. **NOTE:** Look at functions `pinMode()` and `digitalWrite()`.
2. Send the temperature value to the smartphone using Bluetooth or Wi-Fi. **NOTE:** It is recommended to recover parts of the code of the Wi-Fi and Bluetooth sketches.
3. Repeat exercise 1 with the LED in another Arduino linked with Xbee communication. The emitter will send the temperature value and the receiver will decide when to turn ON the LED. **NOTE:** It is recommended to recover parts of the code of the Xbee sketches