TRABAJO FIN DE GRADO

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

MICROCONTROLLER-BASED WIRELESS PEDOMETER PROTOTYPE DESIGN

Curso 2018/2019

Alumno/a:

Ramón Huesa Amat

Director/es:

José Luis Blanco Claraco

UNIVERSIDAD DE ALMERÍA

TECHNICAL PAPER. MICROCONTROLLER-BASED WIRELESS PEDOMETER PROTOTYPE DESIGN

Ramón Huesa Amat

31/05/2019

GENERAL CONTENT

DOCUMENT 1. REPORT

CONTENT

CHAPTER 1. INTERREST AND OBJECT

1 Interest

Pedometers have been used for years by amateur athletes, as well as other people concerned about their health who want to measure their physical activity and their Total Daily Energy Expenditure (TDEE), which is highly influenced by the overall number of steps taken per day [1].

The part of the caloric rate that is independent from exercise can be divided into three main components:

Basal Metabolic Rate (BMR), which is the energy needed for basic body processes such as vital organ functions.

Non-exercise Activity Thermogenesis (NEAT), represents the calories that are burnt from all the movements done during the day that are not considered exercise.

Thermic Effect of Food (TEF). Food digestion is a process that needs energy. Certain food burns more calories than other. The TEF is a percentage of eaten calories.

NEAT is the component that can be influenced the easiest. Therefore, measuring this activity combined with an adequate diet is an interesting tool for controlling one's body fat percentage and thus preventing obesity, which is one of the main health issues in the developed world. The pedometer can be a useful device in that matter, as it seems to encourage people to do more walking [2].

The benefits of physical activity depend on three elements: intensity, duration and frequency of exercise.

Comparing two basic kinds of exercises such as running and walking, it is easily appreciable that walking frequency and duration must be increased so its lower intensity is balanced. That makes running seem much more time-efficient but if transitions such as the extra warm-up, cool-down and changes of clothing and shoes that runners need are weighted, the time difference narrows considerably. If risk of injury is also considered, walking must be chosen as the best option for the average person as highly combinable with people life style as well.

2 Object

Nowadays, there are plenty of different models of pedometers, some of those being embedded in more complex devices such as mobile phones.

The goal of this project has been to develop a pedometer prototype composed of two separated devices with wireless communication. The first one is hooked to the hip at the border of the user's trousers, skirt or any other garment with a similar top border. This device is called the Measurement Portable Device (MPD) and it will be capable of detecting steps taken by the carrier. MPD sends data wirelessly to the other device whose name is User Interactive Displayer (UID), the UID is an interactive display with buttons that makes the user capable of selecting the specific data desired to be displayed, carried in the user's pocket so it can be checked by the user in one hand showing the information on the display.

- The MPD comprises an Arduino microcontroller board, accelerometer, wireless communication system, direct tension supply, and protective and connective parts.
- The UID comprises an Arduino microcontroller board, display, wireless communication, direct tension supply, and protective and connective parts.

This project is built on an Arduino board due to its easy programmability and because, as a prototype, Arduino boards are ready to be programmed and that hasten the overall process. Also due to the extended use of Arduino, it is easy to find a wide variety of tutorials and open-source software and hardware, which makes learning much easier for beginners.

It is important to understand that this project contains not only the manual to implement an application of the Arduino board, the other peripherals and the computer software provides, but also an introduction to microcontroller programming and automatic systems, which may be made extensive to a wider range of possibilities for other engineering projects.

3 System overview

The basis of the system is the MPU, which works detecting steps and sending information to the displaying system. The MPD is designed to communicate in two different modes attending to the way in which this information from the MPD is displayed:

In one of these modes, the MPD communicates with the PC and the information is processed and displayed with Matlab (Figure 1).

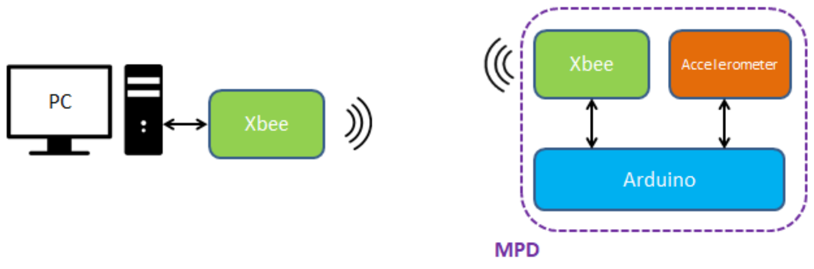


Figure 1. MPD communication with the PC.

In the other mode, the MPD communicates with the UID. The UID displays the information on the OLED display (Figure 2).

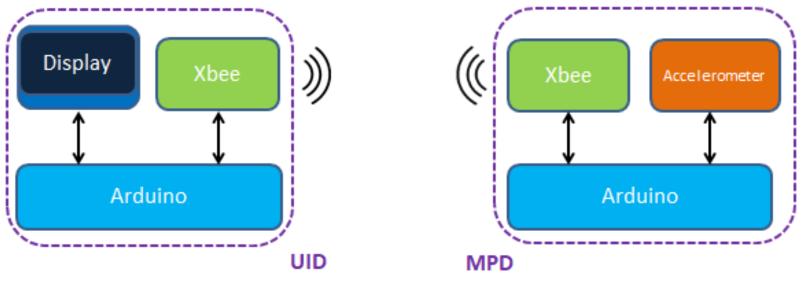


Figure 2. MPD communication with the UID.

In both cases there is one-way communication from the MPD to rather the PC or the UID.

CHAPTER 2. COMPONENTS

This chapter will introduce the components used in the hardware of both devices (MPD and UID). The overall working systems of every component will be explained as well as its configuration method in order to expose not only the current function in this project but also other possible alternatives for future changes.

1 ARDUINO BOARD

Arduino board is the chosen microcontroller for this project for three main reasons:

- Easy programming environment
- Large amount of learning resources available online (tutorials, forums and example sketches).
- Fast starting with Arduino boards which do not require any extra components but the PC (with IDE Arduino free software installed), USB cable and the board.

1.1 Microcontroller

Most Arduino boards are built on AVR microcontrollers Atmel-8 bits (ATmega8, ATmega168, ATmega328, ATmega1280, and ATmega2560), each model is featured by different memory size, number of pins and functions. For each microprocessor there are several boards in the market fitting different resources requirements.

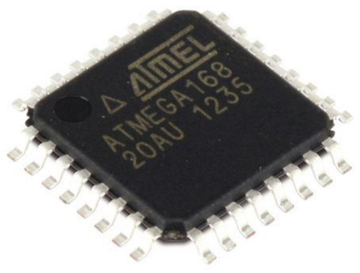


Figure 3. ATmega 168 chip [3].

The microprocessor used in this design is ATmega168 (Figure 3). Arduino Nano also offers the ATmega328 but due to the little memory required in this processing, ATmega168 smaller memory size is not a limiting factor.

1.2 Board

In the market found different board models by Arduino can be. Some of the most popular ones are Mini, Nano, UNO and Mega. Each of them expand their size, functions, pin numbers and memory size to adapt to more demanding projects. Because of the power consumption, size and price, the most adequate board for a project is the smaller one that complies with the requirements for its purpose.

We will be using Arduino Nano (figure 4) as the microcontroller in this project due to its small size and the low amount of pins needed for the design.

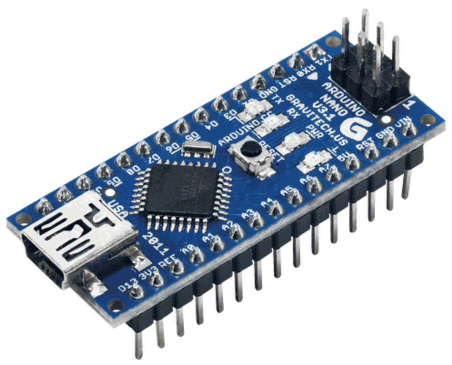


Figure 4. Arduino Nano V3.1 board [4].

1.3 Arduino Nano Characteristics

The Characteristics of the Arduino Nano microcontroller are further explained in their official website [5]. For this project, it necessary to highlight the points that follow:

The ATmega168p used in this project has the following processor features:

Program Memory Type	Flash
Program Memory Size	16KB
CPU Speed (MIPS/DMIPS)	20
SRAM Bytes	1,024
Data EEPROM/HEF	512 bytes
Digital Communication Peripherals	1-UART, 2-SPI, 1-I2C
Capture/Compare/PWM Peripherals	1 Input Capture, 1 CCP, 6PWM
Timers	2 x 8-bit, 1 x 16-bit
Number of Comparators	1
Temperature Range	-40 to 85 ºC
Operating Voltage Range	1.8 to 5.5 V
Pin Count	30

Table 1. Arduino Nano characteristics.

1.3.1 Processor

The Atmel ATmega48/88/168 is a low-power CMOS 8-bit microcontroller based on the AVR-8 architecture [6]. Figure 5 shows the AVR-8 architecture.

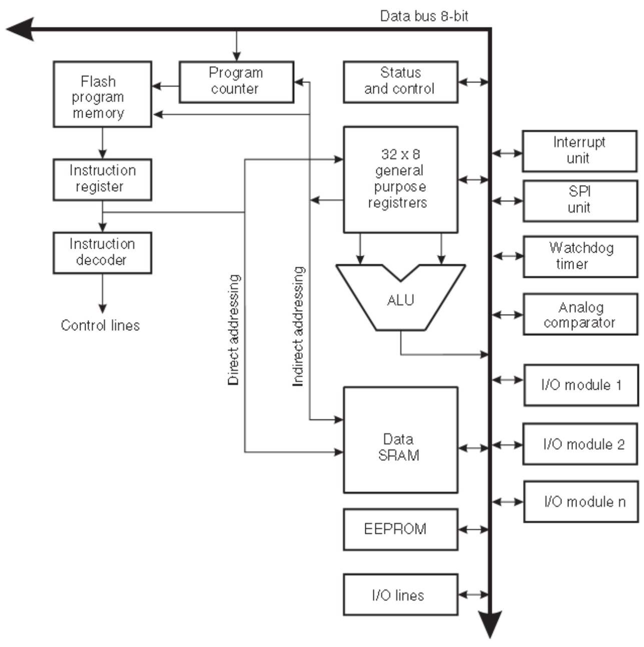


Figure 5. Block diagram of Arduino Nano AVR-8 architecture [6].

The Harvard architecture stores instructions and data in separate memory units that are connected by different busses. There are at least two memory address spaces to work with, so there is a memory register for machine instructions and another memory register for data. Controllers designed with the Harvard architecture are able to run a program and access data independently, and therefore simultaneously [7].

AVR is based in modified Harvard architecture. This variation of the Harvard computer architecture allows the contents of the instruction memory to be accessed as if it were data.

Board clock speed is 16MHz from the oscillator whereas ATmega48P/88P/168P/328 varies in the range of 0 - 20 MHz when the power supply is 2.7 - 5.5V.

1.3.2 Power

Arduino Nano has 3 options for power supply:

- Mini-B USB connection
- Pin 30, 6-20V unregulated (7-12V recommended)
- Pin 27, 5V regulated

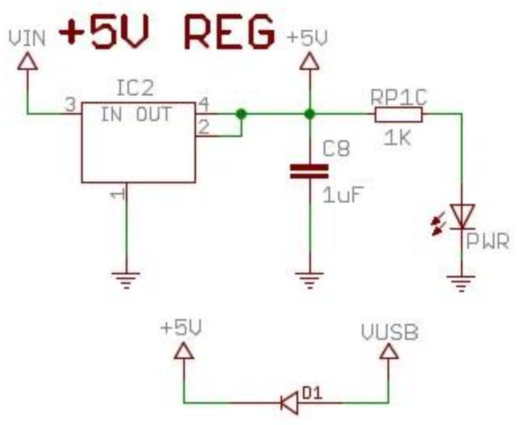


Figure 6. Arduino Nano Schematic [8].

As figure 6 shows, the voltage from Vin is converted into 5V connected to the +5V pin, this pin can be used both as input and as output. Pin +5V is connected to USB as well.

Arduino power consumption is 19mA if no other peripheral is fed by the board.

1.3.3 Memory

Arduino as others AVR microcontrollers from Atmel have 3 different embedded memories: Flash, SRAM and EEPROM [9]

• **SRAM** (static random access memory): Used for local variables and partial data. It is the memory zone used by the sketch to create and manipulate the variables when it is executed. It is a limited resource and it must be supervised as not to be exhausted.

• **EEPROM:** Non-volatile memory used to keep data after a reset. It can be saved from the microcontroller program, usually as “program constant”.

This memory is slightly slower than SRAM and it is more difficult to be used.

• **Flash:** It is the program memory where the already compiled sketch is saved. The bootloader is also saved here which will enable the PC to write on this memory. The program is executed from the Flash memory but data cannot be changed there. They must be copied to the SRAM in order to modify them. As the EEPROM, Flash memory is not volatile and the information persists after turning the Arduino off.

1.3.4 Hardware Connection

The Arduino Nano has 8 analog inputs and 14 digital I/O pins, as well as other 8 pins that cannot be configured in the same way as the others (Schematics in Figure 7).

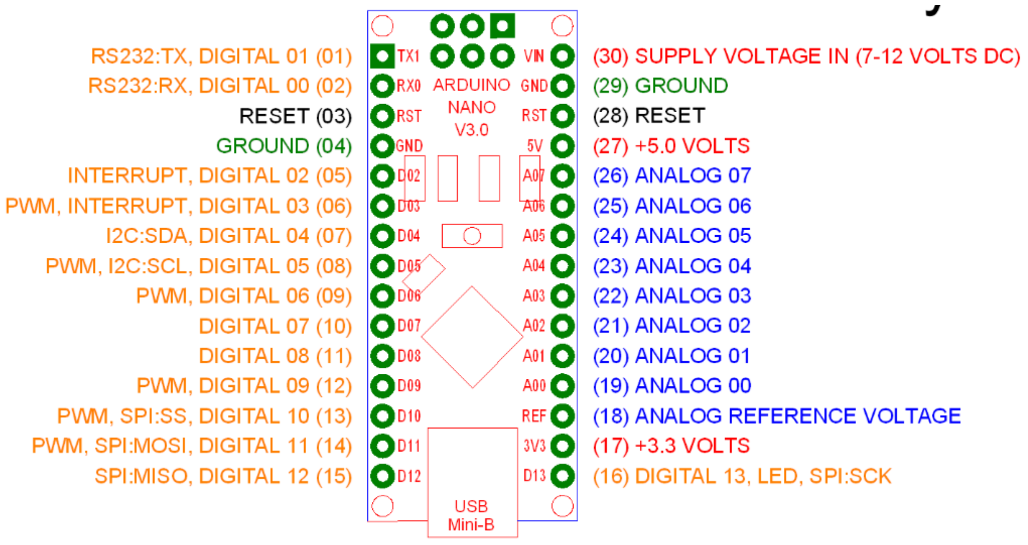


Figure 7. Arduino Nano V3 pins schematic [10].

Digital pins

Arduino has 14 digital pins numbered from 0 to 13 (Figure 5, orange) which can be set rather as inputs or outputs. Each pin can provide and receive a maximum of 40mA (20mA is the recommended maximum) and operates at 5V. They also have an internal pull-up resistor of 20-50kΩ. Some of these pins have also a specialised function:

Serial: Pin 0 (Rx) and 1 (Tx) are used to receive and transmit TTL serial data. These pins are connected to the FTDI USB-to-TTL Serial chip, in such a way, that they share the same channel with the USB port.

External interrupts: Pins 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

PWM: Pins 3 to 6 and pins 9 to 11 provide an 8-bit digital signal oscillating at high speed low and high so this output behaviour can resemble an analog signal. This technique is known as PWM (pulse-width modulation).

SPI: Pins 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK) support SPI communication, which despite not being included in the Arduino language, is provided by the underlying hardware.

LED: Pin 13 is connected to a built-in LED. This LED is on when pin 13 is high and it is off when pin is low.

Analog pins

Nano has 8 analog pins (Figure 5, blue) operating just as inputs, each of which internally convert the signal into 10-bit digital signal. By default they measure from ground to 5V but that can be easily configured to another subrange in between these limits by using AREF pin. They also can be used as digital pins apart from the exception of analog pins 6 and 7. Some of these pins have also a specialised function:

I2C: Pins A4 (SDA) and A5 (SCL) support I2C communication using *Wire* library.

Other pins

Other 8 pins with other functions can be found on the board:

AREF: Reference voltage for the analog inputs.

Reset: 2 pins that will reset microcontroller when any of them is brought to low state.

Power: 7-12V recommended input voltage (lower supply is accepted reducing functionality).

Voltage source: 2 pins with 5V and 3.3V output voltage.

Ground: 2 ground pins used as the 0V reference.

1.3.5 Supported Communication Protocols

Arduino Nano has various facilities to communicate with a computer or another microcontroller:

UART: The ATmega microcontroller provides UART TTL serial communication available on digital pins 0 (Rx) and 1 (Tx).

USB: Arduino Nano can communicate through the USB as COM port channelled by an FTDI FT232RL on the board and a FTDI driver in Arduino software. A serial monitor is also included in the software, which allow sending and receiving textual data from the Arduino board.

LEDs: The Rx and Tx LEDs on the board will flash when FTDI chip sends data through the USB connection to the computer.

Serial: The SoftwareSerial() library allows for serial communication in any of the Nano's digital pins (later used in this project code). For the MPD, the Arduino use digital pins 5 and 6 to contact in Serial Mode to the XBee using SoftwareSerial library. It is explained in detail on chapter 3.

I2C: The ATmega also support a I2C bus communication easy to be used with Wire() library include in Arduino software

SPI: SPI communication is also included in ATmega, every Arduino model support SPI communication in different pins in the Arduino Nano SPI is used in digital pin from 10 to 13.

Arduino SPI.h library allows an easy use, and SCK, MOSI, MISO and SS are included as constant so the code can be interchangeable between different Arduino models.

1.1.1 Programming

The Arduino Nano can be programmed in C++ with Arduino IDE. The ATmega microprocessor in the board comes preburned with the bootloader that allows the user to upload new code without any external programmer hardware.

The original STK500 protocol can be used to communicate. The bootloader can be bypassed and the microcontroller programed through the ICSP (In-Circuit Serial Programming) header using Arduino ISP.

2 MPU-6050 ACCELEROMETER+GYRO

The accelerometer model used to detect the movement of the user is MPU-6050 Accelerometer + Gyro made by InvenSence (figure 8).

The MPU-6050 sensor contains three MEMS accelerometer and three MEMS gyroscopes in a single chip. The analog-to-digital conversion hardware contains a 16-bit signal for each channel. X, Y and Z axes are captures in different channels simultaneously [11].

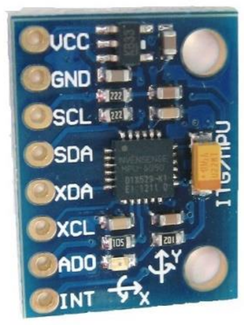


Figure 8. GY-521 breakout board [11].

2.1 Breakout boards

In 2014 the last breakout board version used in this project appeared without any name or code. This is almost equal to a previous version called GY-521 board.

The board has a voltage regulator on the board for 3.3V, two 10k pull-up resistors for the SCL and SDA, 300 ohm resistors in the SCL and SDA signal. The MPU-6050 core requires 3.3V but due to the embedded voltage regulator the board must be connected to 5V to the Vcc.

2.1.1 Hardware Connection

Vcc: pin used for powering the MPU-6050 module with 5V.

GND: is ground pin.

SDA: SDA pin is used for I2C communication data.

SCL: SCL pin is used for I2C communication clock.

XDA: This is the sensor I2C SDA data line for configuring and reading from external sensors.

XCL: This is sensor I2C SCL clock line for configuring and reading from external sensors.

ADO: I2C Slave Address LSB. The pin ADO selects between I2C address 0x68 and 0x69. (3.3V).

INT: Interrupt pin for indication of data ready.

For this Project just pins Vcc, GND, SDA and SCL will be used to connect to the Arduino Nano.

2.2 Characteristics

MPU-6050 datasheet provides the information from the core characteristics [12].

Parameter	Rating
Supply Voltage	-0.5V to +6V
Input voltage Level	-0.5V to VDD + 0.5V
Acceleration	10,000g for 0.2ms
Operating Temperature Range	-40°C to +105°C
Storage Temperature Range	-40°C to +125°C
Output rate	1 kHz

Table 2. MPU-6050 processor characteristics.

2.3 Programming

The accelerometer and gyro readings are called “raw” values, following the standards in the rest of accelerometers and gyro sensors. MPU-6050 has a Digital Motion Processor (DMP) which can be used in more sophisticated applications computering values from the sensor before sending them to the microcontroller.

The DMP can be programmed to do complex calculations with the values from the sensor reducing this way the load for the Arduino. Although InverSense does not supply detailed information how to program the DMP, for this reason in this project the Arduino will just communicate with MPU-6050 with a basic programme to read raw values.

The code from the program is based on the example showed on the Arduino website just by adapting the necessary parts to read the accelerometer and gyro raw values and send them via I2C. The sleep mode must be disabled so the registers for the accelerometer and gyro can be read This is done using Wire() library and writing on the specific register.

2.3.1 Wire() Library

This library allows you to communicate with I2C / TWI devices. Arduino Nano has 2 I2C communication pins used to connect MPU-6050 (Digital pin 4, SDA; digital pin 5, SCL) [12].

There are both 7- and 8-bit versions of I2C addresses. 7 bits identify the device, and the eighth bit determines if it is being written to or read from. The Wire library uses 7 bit addresses throughout so if the in 8 bit codes the low bit must be, yielding an address between 8 and 127 as the addresses from 0 to 7 are reserved. It is important to note that a pull-up resistor is needed when connecting SDA and SCL pins. However Arduino has its own internal pull-up resistors so for this project it will not be necessary to use extra resistors.

The Wire() library implementation uses a 32 byte buffer, therefore, any communication should be within this limit. Exceeding bytes in a single transmission will just be dropped.

2.3.2 Advance Programing Possibilities

The MPU-6050 has much further possibilities and despite not being used for this project, it is important to bear them in mind for possible future modifications:

The sensor can be also programmed to place values in the 1024 byte FIFO buffer so the Arduino can read them from there. The MPU-6050 place data in the FIFO buffer, after which it signals the Arduino with the interrupt signal so the Arduino knows there is data waiting in the buffer to be read.

The MPU-6050 acts as a slave to the Arduino with the SDA and SCL pins connected to the I2C-bus. In addition, the MPU-6050 plays as the master to use a second (sub)-I2C-bus using its own I2C controller. The pins used for this second (sub)-I2C-bus are AUX_DA and AUX-CL.

2.4 Code

Arduino website provides some example sketches to work fast with the MPU-6050 without deepening much into the hardware or communication protocol. The code to control the MPU-6050 in this project is based on the “Short Code Sketch” on the Arduino website using just raw values and omitting DMP [11]. The Arduino code referred to here can be found in “Annex 1. Code”.

3 XBEE

XBee is a programmable module used for wireless connection created by Digi International. XBee modules have their own communication protocol by radio frequency and one of their main advantages is their toughness and the low consumption [13].

3.1 Models

There are different models called series, this are the most widely used: XBee Series 1 (also called XBee 802.15.4), XBee Znet 2.5 (formally called Series 2), ZB (this is the latest Series 2), 2B (even latest Series 2), 900MHz and XSC. In this project I will use series 1 (S1)(figure 9), which is the easiest one to work with and needs very little configuration or none at all if the default configuration is convenient for the target purpose. XBee Series are not compatible among them [14].

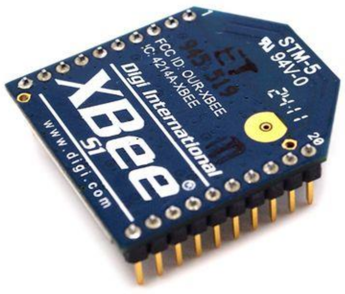


Figure 9. XBee S1 [15]

Another interesting specification from the manual is that XBee operate with just 45-50 mA when is transmitting or receiving data, while XBee is in sleep mode is <10uA, which is negligible. Such a low power consumption is a significant advantage to be chosen for this project's communication stage.

3.1.1 Hardware Connection

XBee has 20 Physical pins just 4 of which will be used for this project figure 10):

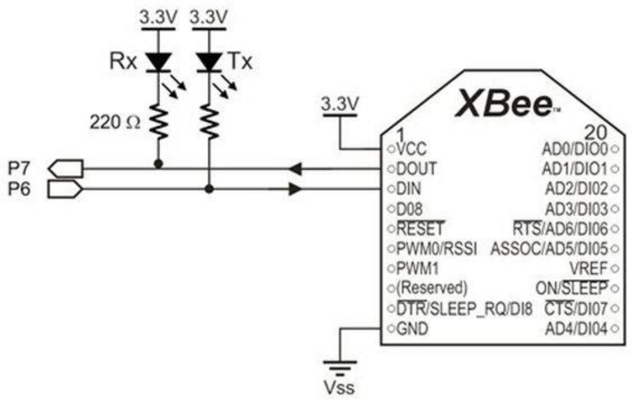


Figure 10. XBee Pins Schematic.

- **Pin 1.** Vcc is the power supply which, in XBee modules is 3.3V.
- **Pin 2 and 3**. DOUT and DIN are the communication pins. Data in and Data out so they both should be connected to the Arduino board at the specific pins set in the code for this serial communication. In this project the chosen pins in Arduino board for this data transmission will be digital pins 5 and 6.
- **Pin 10.** GND is ground. It should be connected to ground pin on the Arduino board.

Pin 2 and 3 must be connected with pull-up resistors so when they are not receiving or sending data they do not have a floating state but instead they are high and protocol is not affected by fault values. These pull-up resistors are implemented in the Arduino board; however, its value can be modified adding two additional resistors.

3.2 Communication protocol

XBee communicates using radio transmission, the standard follows is 802.15.4. XBee can be configured in 2 different modes to communicate: AT (Transparent) and API (Application Programming Interface):

AT Mode

- This mode is useful when it is not necessary to change destination addresses very often, or there is a very simple network or simple point to point communication.
- In AT mode, any data sent to the XBee module is immediately sent to the remote module identified by the Destination Address in memory.
- No packet formation is necessary, so it just sends Serial data to the **Tx** of one XBee and it will be received by the **Rx** of the destination XBee. This communication procedure is also known as Transparent Mode.

API Mode

- For larger networks that involve nodes talking to multiple targets, API mode can be more convenient.
- Data must be formatted in frames with Destination information and payload
- API mode allows to change destination address much more quickly because Command Mode does not need to be entered. [18]

As in this project the communication network is composed of just 2 XBee modules, AT is enough to allow good communication and for the project this is the chosen mode.

3.3 XBee USB

XBee models also include a version integrated with a USB male port protected in a more solid shell to be directly connected to the PC (figure 11). It is configured in the same way as the XBee but the communication with the PC is through USB instead of the I2C bus

In this project the XBee USB is used for the test with the MDP to develop the algorithm as well as to test the final mode substituting the UID. Matlab is be the software designed to receive and process the data from the COM port which the XBee USB is connected to.

Figure 11. XBee USB 802.15.4 [16]

3.4 Adaptor Module

XBee's pins are distributed in 2mm width, slightly smaller than common boards, therefor an adaptor module will often be require. These adaptor modules may have a power convertor from 5V to 3.3V so we can still use a power supplier of 5V for the overall project and not to be afraid of damaging the XBee.

The adaptor used in this project is made by Roarkit (figure 12). XBee must be connected to 3.3V as the adaptor does not convert voltage. By using this adaptor module the board allows manually coupling and removing the XBee module since it is UAL property and must be turned in after the assessment of this project.

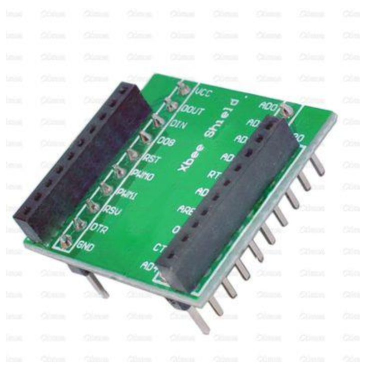


Figure 12. Adaptator module for XBee [17].

4 0.96’ OLED Display

This section explains the characteristics of the 0.96 inch monochrome OLED display from Geekcreit (figure 13), how to connect to the Arduino and use it to show the information obtained from the MPD.

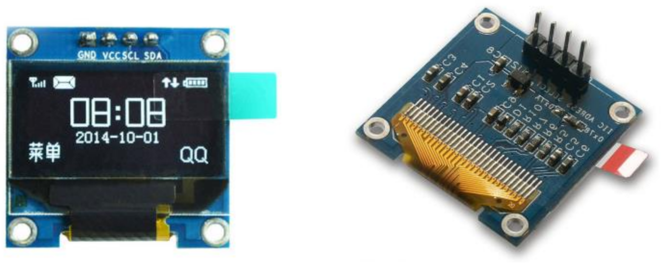


Figure 13. 0.96" OLED display [20].

4.1 Characteristics

This small OLED display is characterised by its low consumption, good luminosity, wide vision angle and relatively good resolution [19].

Resolution	128x64 pixels
Vision angle	>160º
Working voltage	3 - 5V
Interface	I2C
IC Driver	SSD1306
Working temperature	-30ºC - 70ºC
Colour	White
I/O Pins Voltage	3V3 and 5V
Dimensions	27x27x4 mm

Table 3. OLED display characterisitics.

4.1.1 Hardware Connection

The display connects to Arduino using four wires – two for power and two for data. The data connection is I2C (I²C, IIC or Inter-Integrated Circuit) [20].

At the very lowest level, the Arduino Wire library is used to communicate with the display. Libraries are available that make it easy to start using the display right away to display text and graphics.

These are the connections from the Geekcreit 0.96 Inch 4 pin I2C OLED module to Arduino:

- Vcc pin from OLED to the 5V pin from the Arduino for powering the display.
- GND from OLED to GND from the Arduino.
- SDA from OLED to SDA (A4, physical pin 27) from the Arduino.
- SCL from OLED to SCL (A5, physical pin 28) from the Arduino.

Important to note is that some of the displays may have the GND and VCC power pins swapped around (GND VCC SDA SCL

4.2 Arduino libraries

Two Arduino libraries must be installed to start using the display. The SSD1306 driver library is used to initialize the display and provide low level display functions. The GFX library provides graphics functions for displaying text, drawing lines and circles, etc. Both libraries are available from Adafruit website.

Another library that can be useful is “Adafruit_GFX.h” that provides a common syntax set of graphics functions for all the LCD and OLED displays Arduino compatible. This allows Arduino sketches to easily be adapted between display types with minimal fuss. This library is included in the code but there are no syntax of it.

4.3 Programming

Programing the OLED display is an easy task provided that the SSD1306 driver library is used. The OLED display is entirely controlled with commands from the library. The OLED display must be begun and some set up be done, after that plain text can be sent through display.print() command to the screen and will be displayed after the display.display() line [22].

Read “Annex 1. Code” to see the code used for the OLED 0.96 display.

5 POWER SUPPLY

This project must be powered with direct 5V tension by using a lithium battery of 3.7V combined with the conversion module which it is connected to. Both devices also include protection systems to avoid overcurrent, over charge and over discharge.

5.1 LITHIUM-ION BATTERY

A lithium-ion is a type of rechargeable battery commonly used for portable electronics and electric vehicles.

In the batteries lithium ions move from the negative electrode to the positive electrode during discharge and back when charging. Non-rechargeable batteries use metallic as one electrode material whereas Li-ion batteries use an intercalated lithium compound. The batteries have a high energy density, no memory effect and low self-discharge. They can however be a safety hazard since they contain a flammable electrolyte, and if damaged or incorrectly charged can lead to explosions or fires.

The battery used for this project is a 3.7v 1000mAh lithium-ion polymer rechargeable battery made by YCDC (figure 14). Despite de fact that the average voltage is lower than the Arduino required voltage the Conversion Module will enforce this conversion.

The built-in smart protective PCB charging module, also prevents over charging, over discharging, over current and short circuit.

Figure 14. 3.7V Lithium Battery by YCDY

5.2 Features

Model	803040
Type	Lithium battery
Capacity	1000mAh
Rechargeable	YES
Charging Voltage	4.2V
Rated Voltage	3.7V
Working Temperature	-10°~50°
Dimensions (LxWxH)	40x30x8mm

Table 4. Lithum Baterry characteristics.

5.3 Manufacturer's Recommendations

- The user must consider small deviation from the specifications due to human measurements.
- When the battery is not in use, it must be removed from the device.
- It is important to make sure that there is no connection between the red and black wires at any time to avoid short circuit after the battery has been charged.

Lithium must be kept protected from hits, violent movements, excessive heat or any trigger which can increase risk of fire or explosion. That is the reason why the whole system is protected inside a plastic box and the battery glued to its bottom.

6 TP4056 CHARGER MODULE

In order to protect the lithium battery against unexpected electric conditions and in order to allow easy charging, the battery will not be connected directly to the Arduino but to a Charger Module.

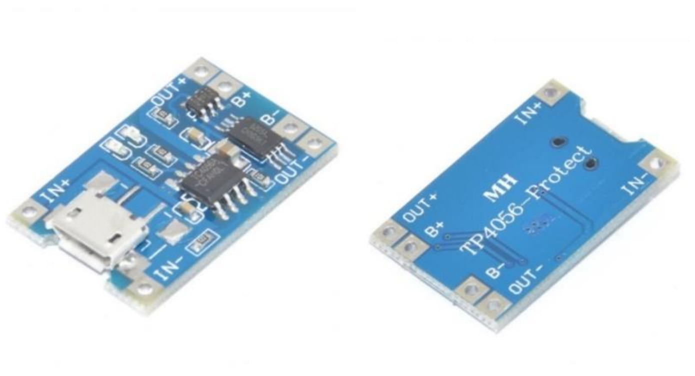


Figure 15. TP4056 Charger Module [21].

The Charger Module used for this project is the Smart Electronics 5 V TP4056 micro USB 1A lithium battery charging board with protection charger module for Arduino (Figure 15) [24, 25].

The module is designed to protect the battery and simplify charging through a micro USB port compatible with many phone chargers. If the phone charger provides 1A, the charging period is approximately an hour.

To charge the battery it must be connected to the module and the charger inserted into the USB module port, the red light signal that the battery is being charged until the green light when it is full.

6.1 Hardware Connection

The module has 3 pairs of connections and the USB:

- **B+/B-:** Here the battery wires are connected. Red wire must be connected to B+ and black wire to B-.
- **OUT+/OUT-:** connected to the load, in this case the Arduino board. It provides the voltage supplied by the battery. OUT+ must be connected to Vcc and OUT- to GROUND, otherwise it may damage the Arduino.
- **Micro USB:** Female Micro USB can be directly input to the phone charger with 5V tension.
- **IN+/IN-:** This is the alternative and convenience DIY input for future charging method. It perform the same function as the Micro USB but it can be soldered to a different port or device and receive 5V as well (not used in the project).

6.2 Characteristics

Input voltage (min - typ - max)	4.0 – 5.0 – 8.0V
Charging cut-off voltage	4.2V ± 1%
Maximum charge current	1000mA
Battery over-discharge protection	2.5V
Battery overcurrent protection	3A
Board size	2.6x1.7 cm

Table 5. Charger Module characteristics.

Note that the charger used to connect the module must be able to output at least 1A or it may not charge.

7 Other components

The following components are not very complex but must be briefly explained as well.

7.1 Toggle Switch Handle

The toggle switch handle (figure 16) opens the circuit and switches the system off while it is not used. It has 3 pins and the switch connect the middle one with the one on the side where the handle is shifted.

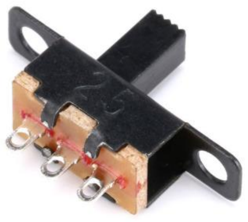


Figure 16. Toggle Switch Handle.

7.2 External Protection

The whole system is vulnerable to be touched by other objects, water and other elements from the environment. The most vulnerable parts are the lithium battery and the assembling soldering so it must be protected inside the protecting shell in order to make it easier to be manipulated and to extend its life expectancy.

The protection is a white plastic box with measures of 70 X 45 X 30mm (figure 17), 30mm is may however be too high for our project so in the assembling half of the box is cut off.

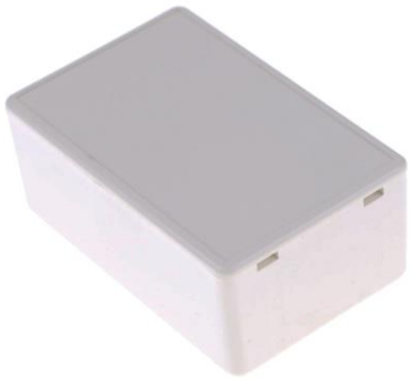


Figure 17.Case Enclosure Box.

7.3 Buckle Webbing Ending Clip

The protection shell of the MPD also has the couple system to fasten the device to the top border of the lower body garment (tests and algorithm are developed using the trousers´ belt to couple the MPU-6050to the user and this method will be the most trustworthy one to obtain correct readings from the sensor). The buckle webbing ending clips (figure 18) are cut and glued to the box, explained in Annex 2. Assembling.

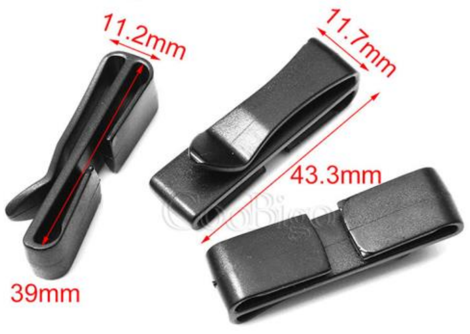


Figure 18. Buckle Webbing Ending Clip [22].

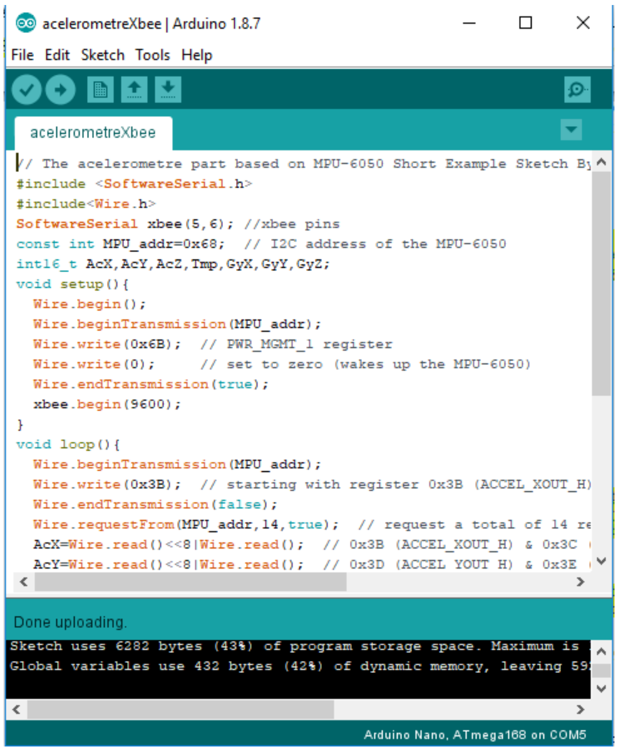
CHAPTER 3. SOFTWARE

1 IDE Arduino

The computer software used in this project to program the microcontroller is IDE Arduino. It is an easy-to-use editor and compiler by Arduino Genuino which makes creating a program so easy that even students with very little knowledge of electronic or information technology can use it.

1.1 User Interface

Figure 19 shows the characteristic IDE user interface is. It does not have many options as it is designed to be simple and easy to use.



```
// The acelerometre part based on MPU-6050 Short Example Sketch By
#include <SoftwareSerial.h>
#include<Wire.h>
SoftwareSerial xbee(5,6); //xbee pins
const int MPU_addr=0x68;  // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
void setup(){
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);  // PWR_MGMT_1 register
  Wire.write(0);     // set to zero (wakes up the MPU-6050)
  Wire.endTransmission(true);
  xbee.begin(9600);
}
void loop(){
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 re
  AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C
  AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E
```

Done uploading.

```
Sketch uses 6282 bytes (43%) of program storage space. Maximum is
Global variables use 432 bytes (42%) of dynamic memory, leaving 59
```

Arduino Nano, ATmega168 on COM5

Figure 19. IDE Arduino user interface

IDE Arduino runs on Windows, Mac OS X and Linux. The environment is written in Java and based on Processing and other open-source software. IDE Arduino (which stands for Integrated Development Environment) uses C++ programing language customised with libraries which transform the language into a higher level language, easier to use for simple project, these code functions are explained on the Arduino website and complemented with great variety of tutorial widely available in internet.

1.2 Program code Schematic

Arduino code structure can be separated in 3 main parts:

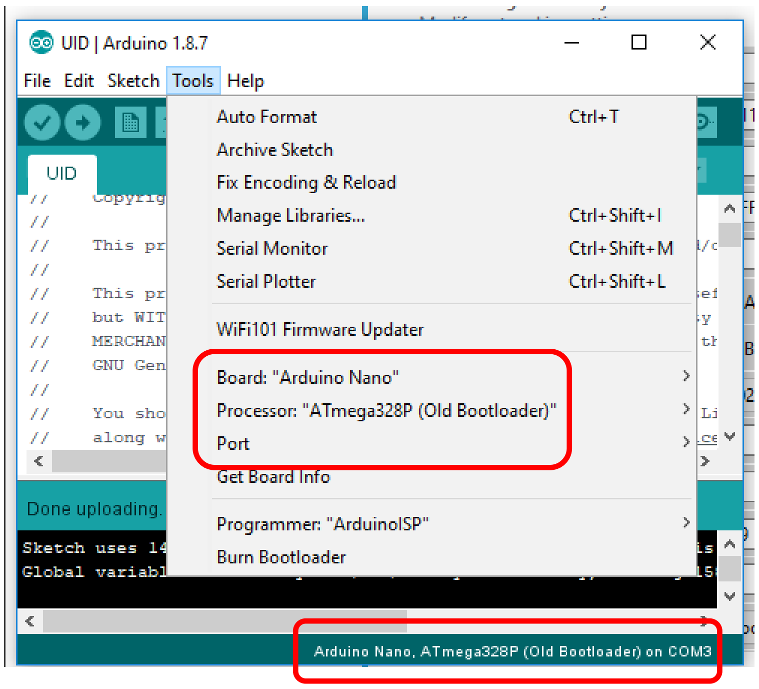
1 - In the head of the code libraries must be used as well as variables and constants.

2 - After that first part we can find the void setup function. In this function all the setup instruction related with the processor should be placed, also other embedded devices with a default setup that must be changed before starting running the rest of the program. For instance, to set the physical pins or registers mode. This part is run every time the Arduino is turn on or reset.

3 - The last part is the main code inside the void loop function which will be run repeatedly in a loop as the processor is powered on.

When the code is written it can be verified with the V symbol on the left top corner showing any error and comments on the bottom black window and if the code is error-free it can be uploaded to the board connected pressing the -> symbol next to the previous one.

1.3 Configuring Arduino IDE



Board: "Arduino Nano"
Processor: "ATmega328P (Old Bootloader)"
Port

Done uploading.

Arduino Nano, ATmega328P (Old Bootloader) on COM3

Figure 20. IDE Interface configuration.

Arduino IDE must be configured to connect to the microcontroller when the code has been compiled and is then going to be saved on it. This configuration is made in Tools options and always appears in the bottom margin of the main window (Figure 20), just 3 parameters have to be set up:

- Board: Arduino Nano.
- Processor: ATmega328P (Old Bootloader). Despite the microcontroller used being 168 this option is the only one acceptable for the program to upload new software, 168 option will call an error.
- Port: COM3. It completely depend of every single computer ports management)

This configuration is made in Tools options and always appears in the bottom margin of the main window.

1.4 MPD Code

Here the code in the Arduino from the MPD will be explained. There are two different codes used for the MPD. The first one, called MPD.Tests.ino is the one used to test the MPD and analyse the reading so as to find the best algorithm to detect steps. The second code, called MPD.ino, is just slightly different and it applies this algorithm to the readings so it detects the steps itself and sends the detected steps to the UID. There is also another branch to transmit to the PC using Matlab to read it called MPD_M.ino. For generic Arduino code the main structure is going to be explained and the whole commented code can be found in the “Annex 1. Code”, which will help to make them understood:

1.4.1 Set up

1 - First of all in Arduino code libraries must be called. All the functions used in the code that belong to a library make programming easier but that library must be included in the top of the code.

2 - After it some variables can be declared giving them a value or not. These variables will be available for the whole code, not as if they were declared inside of a function that would make them only available inside that function.

3 - After that the set up code starts. This part of the code will be run just once before being followed by the loop part which will be running in a loop as long as the Arduino is powered or reset. Here the physical pins which are going to be used are defined as inputs or outputs.

1.4.2 Raw values

This part of the code is based on the model code “Short Code Sketch” made by Arduino on its website to pull raw values from the accelerometer. The first lines are used to disable sleep mode and set the accelerometer to send ray values to the registers. Then those values are read and saved in the Ax, Ay, Az, Gx, Gy and Gz variables so they can later be sent to the XBee.

1.4.3 Software Serial

For the MPD, Arduino will be using physical pins 5 and 6 to contact with serial protocol to the XBee as Tx and Rx using SoftwareSerial library. The Arduino website explains how to use the SoftwareSeral function for Arduino projects [23]

The software serial command is presented as SoftwareSerial(RxPin, TxPin, inverse logic). Rx and Tx must be located in a digital pin from the Arduino. For this prototype the chosen pins are 5 and 6 but any of the other free digital pins could be chosen.

SoftwareSerial is used to create an instance of SoftwareSerial object, whose name needs to be provided, which for this project it is XBee. SoftwareSerial, the program will refer to it through this name as more than one SoftwareSerial object can be created, however only one can be active at a given moment. The inverse logic argument is optional and defaults to false, for this code this argument will not be used and so it will remain false.

The communication is enabled when SoftwareSerial.begin() is called.

Parameters

RxPin: This pin is used to receive serial data, for this argument it is just needed the physical number of the pins used.

TxPin: This pin is used to transmit serial data, for this argument the physical number of the pins used is just needed.

Inverse logic: a high value of this argument will invert the sense of incoming bits treating LOW as HIGH and HIGH as LOW. It also affects the way that it writes to the Tx pin. Default value of this argument is False, which is normal logic.

For Serial communication it is important to consider that both devices connected must use the same voltage as receiving high voltage than what the device is prepared for can damage it permanently.

1.4.4 Code for tests

This code will be used to read the values from the MPU-6050 sensor and send them to the PC to be analysed with Matlab. Through this analysis the algorithm for detecting steps will be decided and it will be implemented in the final MPD code.

The code and its explanation can be found in the “Annex 1. Code”.

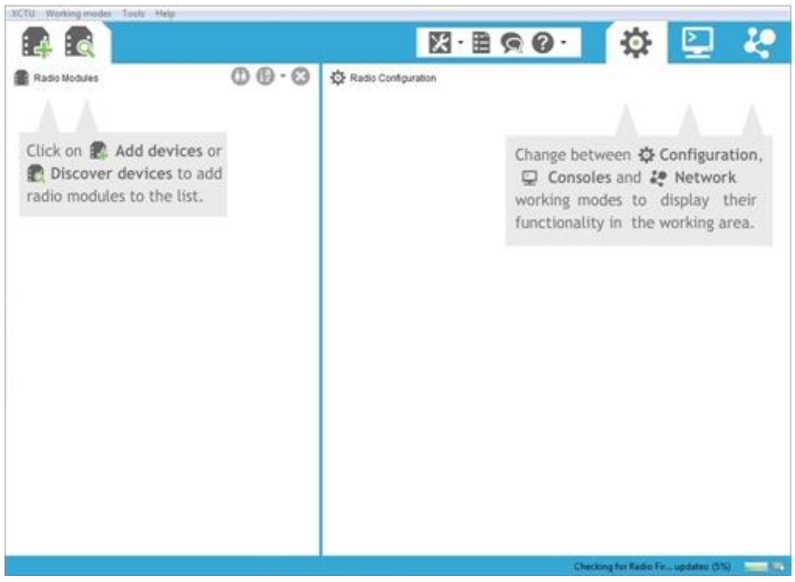
1.4.5 Code for the final MPD

This code is the final program for the prototype and the one used in the physical device developed for this project.

The code and its explanation can be found in the “Annex 1. Code”.

2 XCTU

XCTU is the chosen software to configure XBee modules as it is the recommended one provided by Digi.


XCTU Working modes Tools Help
Radio Modules
Radio Configuration
Checking for Radio Fir... updates (5%)

Click on Add devices or Discover devices to add radio modules to the list.

Change between Configuration, Consoles and Network working modes to display their functionality in the working area.

Figure 21. XCTU main menu. [24]

As Figure 21 shows, XCTU window is divided into five main sections: the menu bar, main toolbar, devices list, working area, and status bar.

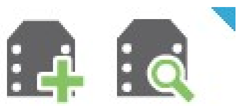


Figure 22. XCTU icon

The main toolbar is located at the top of the application and the first section contains two icons used to add radio modules to the radio modules list (Figure 22).

By clicking on any of them, the user can rather add or discover radio modules connected directly to the computer and add them to your list. Once the module is added it can be configured (in AT mode for this project). A new window with all the configuration settings appears when the module is double clicked, the values are read from the module and now they are ready to be modified.

Modules are configured by default and not many changes should be done.

The most important one for XBee so it is using AT mode is that API mode is disabled.

The first thing that must configure is the Pan ID. The Pan ID is the network address so both XBees need to operate with the exact same one.

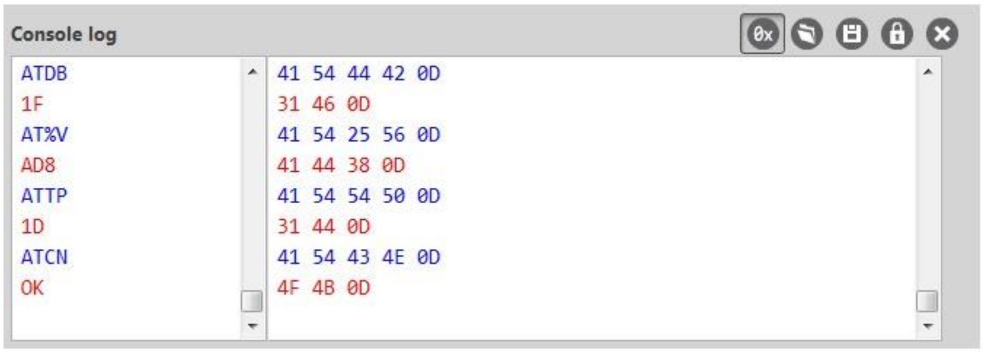
It is important that the destination addresses are configured so both XBee can communicate to each other. Configuration will be explain later on.

One of the modules must be configured as a master while the other one is configured as a slave, to set one XBee as a master the option Coordinator must be enabled. For several modules networks it is important that the master is turned on, otherwise the network will not be working.

After all the changes in the configuration have been set, they must be save in the XBee module by clicking the "Writ*e"* button.

2.1 Console

To check the communication, using the console tool might be very useful (figure 23). It can be found on the top right corner and it works in a similar way as the serial monitor in IDE Arduino.

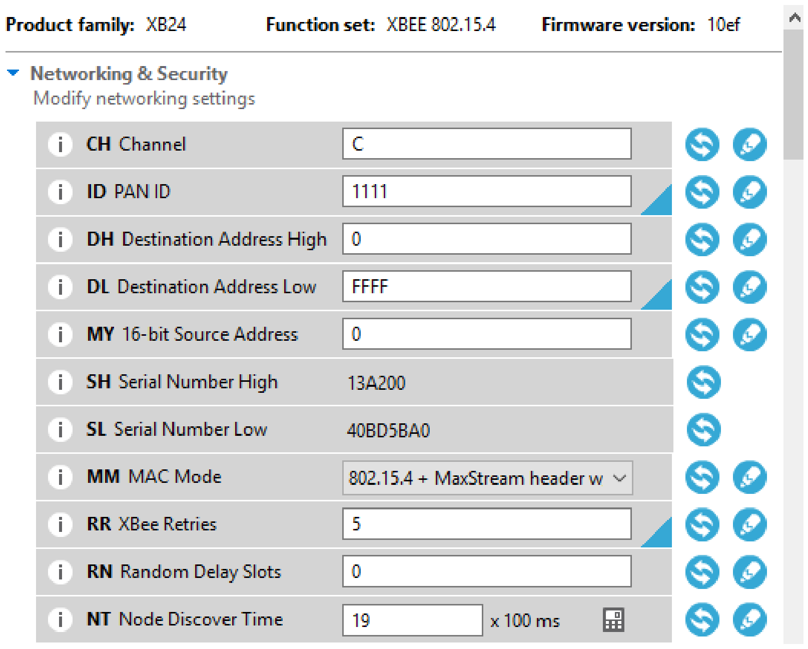


```
ATDB    41 54 44 42 0D
1F      31 46 0D
AT%V    41 54 25 56 0D
AD8     41 44 38 0D
ATTP    41 54 54 50 0D
1D      31 44 0D
ATCN    41 54 43 4E 0D
OK      4F 4B 0D
```

Figure 23. XCTU console log in AT mode.

The console allows the use to type rather in ASCII code on the left as in hexadecimal code on the right simultaneously translated and sent to the XBee. The received data is shown in red font whereas sent data is coloured in blue. With a basic program in Arduino, set to reply when data is received by the XBee, it is easy to test if the XBee communication is working properly. When this test is successful, the XBee is likely to be well configured.

2.2 Configuring the XBee


Product family: XB24 Function set: XBEE 802.15.4 Firmware version: 10ef

Networking & Security
Modify networking settings

CH Channel	C
ID PAN ID	1111
DH Destination Address High	0
DL Destination Address Low	FFFF
MY 16-bit Source Address	0
SH Serial Number High	13A200
SL Serial Number Low	40BD5BA0
MM MAC Mode	802.15.4 + MaxStream header w
RR XBee Retries	5
RN Random Delay Slots	0
NT Node Discover Time	19 x 100 ms

Figure 24. XCTU Configuration Window.

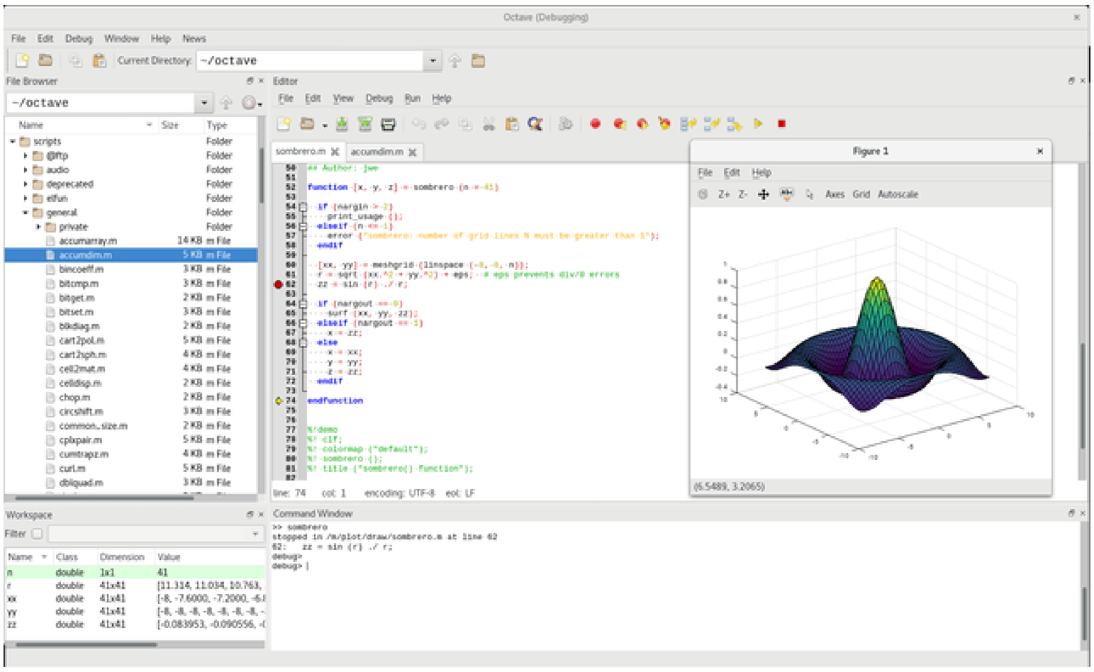
Configure the XBee is simple, when the XBee icon is double clicked the configuration window with the current configuration is displayed (Figure 24). All the values must be set as default by clicking on default bottom if they are not default values and then just the following values have to be changed:

ID PAN ID	1111
DH Destination Address High	0
DL Destination Address Low	FFFF
Coordinator Enable	Coordinator[1]
AP API Enable	API disable[0]

Table 6. XBee registers configuration.

3 Matlab

In order to create the algorithm for the step counting, data must be analysed and the chosen software for this task is Matlab. Matlab offers a powerful platform for data analysing with useful tools as plot, script programs easy access to saved variables. In addition, the software license is provided by Universidad de Almería for all the students.



Name	Size	Type
scripts		Folder
@ftp		Folder
audio		Folder
deprecated		Folder
elfun		Folder
general		Folder
private		Folder
accumarray.m	14 KB	m File
accumdim.m	5 KB	m File
bincoeff.m	3 KB	m File
bitcmp.m	3 KB	m File
bitget.m	2 KB	m File
bitset.m	3 KB	m File
blkdiag.m	2 KB	m File
cart2pol.m	5 KB	m File
cart2sph.m	4 KB	m File
cell2mat.m	4 KB	m File
celldisp.m	2 KB	m File
chop.m	2 KB	m File
circshift.m	3 KB	m File
common_size.m	2 KB	m File
cplxpair.m	5 KB	m File
cumtrapz.m	4 KB	m File
curl.m	5 KB	m File
dblquad.m	3 KB	m File

```
## Author: jwe

function [x, y, z] = sombrero (n = 41)

  if (nargin > 2)
    print_usage ();
  elseif (n <= 1)
    error ("sombrero: number of grid lines N must be greater than 1");
  endif

  [xx, yy] = meshgrid (linspace (-8, 8, n));
  r = sqrt (xx.^2 + yy.^2) + eps;  # eps prevents div/0 errors
  zz = sin (r) ./ r;

  if (nargout == 0)
    surf (xx, yy, zz);
  elseif (nargout == 1)
    x = zz;
  else
    x = xx;
    y = yy;
    z = zz;
  endif

endfunction


%!demo
%! clf;
%! colormap ("default");
%! sombrero ();
%! title ("sombrero() function");
```

line: 74 col: 1 encoding: UTF-8 eol: LF

Name	Class	Dimension	Value
n	double	1x1	41
r	double	41x41	[11.314, 11.034, 10.763,
xx	double	41x41	[-8, -7.6000, -7.2000, -6.
yy	double	41x41	[-8, -8, -8, -8, -8, -8, -8, -
zz	double	41x41	[-0.083953, -0.090556, -

```
>> sombrero
stopped in /m/plot/draw/sombrero.m at line 62
62:    zz = sin (r) ./ r;
debug>
debug> |
```

Figure 25. Matlab User Interface.

In Matlab's window user can find the following spaces (Figure 25):

- Script window
- Command window
- Workspace
- Current folder
- Tools bar

For the tests a script in Matlab is created to analyse the data received in the COM port. Data is visualised on the plot generated when the script is run and it also allows to analyse the variables in the Workspace and operate with them in the Commands Window.

Matlab's system will not be explained in this document. For further explanation consult Matlab user manual or its website https://es.mathworks.com.

3.1 Plotting

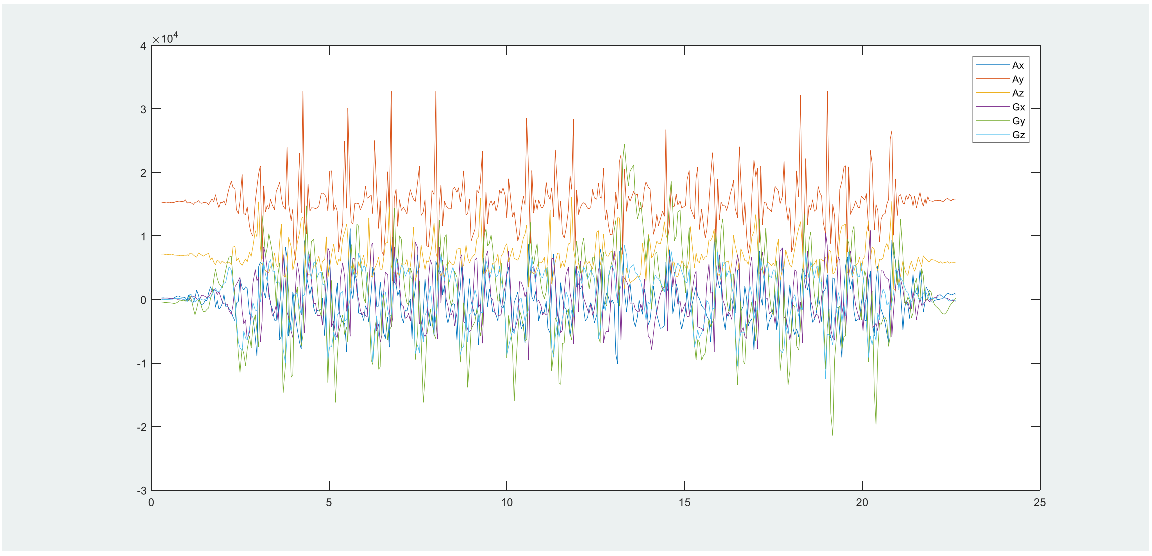


Figure 26. Matlab script plot.

The plot in Figure 26 shows how each step produces a time response with a in the accelerometer axis whose transient state has a short settling time but a significant pic. This pic value depends from one step to another.

3.2 Code

The code used in Matlab consists in a script that converts receiving data into 6 vectors, one for each variable, and also plots it. Through the analysis of the plots and the values in the vector the algorithm is obtained.

The exact code and its explanation can be found in the "Annex 1. Code".

CHAPTER 5. ORGANIZATION CHART AND DEVELOPMENT PHASES

The following table resume the development phases of this project:

- First steps where learning how to program the different components and software to use.
- The Other labour later were to describe and explain in the report all this components and software that were going to be used.
- Creating then the analysis and the rest of information in the project that was a large part.
- Finally revising and correcting the mistakes in the drafting as English is not the author tongue language, this fact made the overall process slower but provided large knowledge to the author .

PHASE	UNDERTAKING	OCTOBER	NOVEMBER	DECEMBER	JANUARY	FEBRUARY	MARCH	APRIL	MAY	DURATION
LEARNIGN PROGRAMMING	ARDUINO	■	■							25
	XBEE		■	■						15
	MPU			■						7
ASSEMBLING	MPD 1.0			■						15
	MPD 2.0				■	■				15
	MPD 3.0							■		12
	UID							■		6
DRAFING	COMPONENTS				■	■		■		30
	SOFTWARE				■	■			■	15
	ANNEXES						■	■	■	25
CODE	MPD			■	■	■	■	■	■	35
	MATLAB					■	■	■	■	30
	UID								■	15
CORRECTIONS AND ADJUSTMENTS								■	■	45
								TOTAL		290

CHAPTER 6.
RESULTS AND CONCLUSIONS

1 Introduction

The aim of this project has been to create a pedometer prototype. Bearing in mind that the commercial product still requires large development work, this process has been successful because the obtained device follow the conditions of a technical project and its hardware and software meet the standards set at the beginning:

2 Hardware

The two devices are well sized and adequately resistant to its actual use. Both are portable and easy to use.

In the MPD, the electronic circuit is not protected as it would make the prototype considerably bigger than it was planned in the previous design version of the MPD. For a user, circuits without cover would be an inconvenience but as a prototype used only for testing purposes it is acceptable.

The hardware design of the UID cannot be constructed for this project but it is a less useful part as the main purpose of this project is the steps measurement and the output can be connected to different kind of interfaces.

3 Software

The actual algorithm of the MPD is basic and still can be improved, still the ability to count steps is now consistent and the error is minimum. However, the current software is the base of future modifications and additions in the code to add more functionalities.

The MPD is capable of measuring slow, medium and fast steps and ignore fast and repetitive movements or just minimise them adding few false steps.

The UID shows real time step count and can be turned on and off without losing counting as it is saved the whole time in the MPD.

4 Development

The purpose of this project is to create a prototype that can be developed till the very end of a final commercialised product or to help in the investigation of other projects in this area of technology. For this aim some possible branches of modifications for the development of the project will be explained for further works on it:

Additional functionalities: the step measurement has been the main goal of this project but, once it is reached, adding information for the user based on their steps count is relatively easy to program. These new features might be maximum speeds, average speed, resting times or other interesting data.

Another possibility of improvement of the code would be to allow the MPD to be coupled in different angles. Now the MPD must be placed with the Y axis pointing perpendicularly to the floor but with some trigonometry transformation it could allow the user to carry it in any position.

The OLED 0.96 display does not allow printing much information on the screen apart from the step count due to its small size so in order to do these kinds of modifications a different display should be used or bottoms added to allow navigation through different windows on the screen.

CHAPTER 7. BIBLIOGRAPHY

[1] J. R. Scott, “Very Well Fit,” [Online]. Available: https://www.verywellfit.com/what-is-energy-expenditure-3496103. [Accessed 16 01 2019].

[2] M. Porter, Interviewee, BBC Rabio 4 - INSIDE HEALTH - Programme 2. [Entrevista]. 02 10 2018.

[3] «Pollin Electronic,» [En línea]. Available: https://www.pollin.de/p/microcontroller-atmel-atmega168-20au-101131. [Last entrance: 24 04 2019].

[4] «Electronics Lab,» [En línea]. Available: http://www.electronics-lab.com/make-your-own-arduino-nano-diy-arduino-nano/. [Last entrance: 24 04 2019].

[5] «Arduino,» [En línea]. Available: https://store.arduino.cc/arduino-nano. [Last entrance: 10 02 2019].

[6] A. Corporation, «ATMEGA168 Datasheet,» 2011.

[7] «Mocrocontrollers Tips,» [En línea]. Available: https://www.microcontrollertips.com/difference-between-von-neumann-and-harvard-architectures/. [Last entrance: 03 05 2019].

[8] [En línea]. Available: https://www.digikey.com/reference-designs/en/open-source-mcu-platforms/2556. [Last entrance: 03 05 2019].

[9] «Aprendiendo Arduino,» [En línea]. Available: https://aprendiendoarduino.wordpress.com/2017/06/21/memoria-flash-sram-y-eeprom-3/. [Last entrance: 29 03 2019].

[10] [En línea]. Available: https://forum.arduino.cc/index.php?topic=243653.0. [Last entrance: 24 04 2019].

[11] «Arduino Playground,» [En línea]. Available: https://playground.arduino.cc/Main/MPU-6050. [Last entrance: 5 3 2019].

[12] InvenSense, MPU-6000 and MPU.6050 Product Specifications Revision 3.4, 2013.

[13] «Arduino Genuino,» [En línea]. Available: https://www.arduino.cc/en/Reference/Wire. [Last entrance: 05 05 2019].

[14] [En línea]. Available: https://geekytheory.com/tutorial-arduino-comenzando-con-xbee/. [Last entrance: 19 01 2019].

[15] [En línea]. Available: https://xbee.cl/que-es-xbee/. [Last entrance: 19 01 2019].

[16] [En línea]. Available: https://www.seeedstudio.com/XBee-PCB-Antenna-S1-802-15--p-1227.html. [Last entrance: 24 04 2019].

[17] «Digikey,» [En línea]. Available: https://www.digikey.com/eewiki/display/Wireless/XBee+AT+Mode+-+Transmit+and+Command+Mode+Example. [Last entrance: 2019 02 01].

[18] «Digi,» [En línea]. Available: https://www.digi.com/products/networking/rf-adapters-modems/xstick. [Last entrance: 24 04 2019].

[19] [En línea]. Available: https://tinkersphere.com/breadboard-perfboard-prototyping/951-xbee-breadboard-adapter.html. [Last entrance: 24 04 2019].

[20] [En línea]. Available: https://www.ledats.pl/en/oled-lcd-display/3813-oled-096-i2c-serial-blue-display-module-lk3.html. [Last entrance: 12 05 2019].

[21] [En línea]. Available: https://diotronic.com/pantalla-oled-0-96-i2c-128x64_32304/. [Last entrance: 28 04 2019].

[22] [En línea]. Available: https://startingelectronics.org/tutorials/arduino/modules/OLED-128x64-I2C-display/. [Last entrance: 28 04 2019].

[23] [En línea]. Available: https://startingelectronics.org/tutorials/arduino/modules/OLED-128x64-I2C-display/. [Last entrance: 10 05 2019].

[24] [En línea]. Available: https://www.addicore.com/TP4056-Charger-and-Protection-Module-p/ad310.htm. [Last entrance: 28 04 2019].

[25] N. T. P. A. Corp., TP4056 - Datasheet.

[26] [En línea]. Available: https://www.aliexpress.com/item/5pcs-3-4-2-Quick-Slip-Keeper-Plastic-Buckle-Webbing-Ending-Clips-adjusting-strap-Belt-molle/32822577828.html?spm=a2g0s.9042311.0.0.27424c4dA6eoXM. [Last entrance: 1 05 2019].

[27] A. Genuino. [En línea]. Available: https://www.arduino.cc/en/Reference/SoftwareSerialConstructor. [Last entrance: 25 03 2019].

[28] «Digi,» [En línea]. Available: https://www.digi.com/resources/documentation/digidocs/90001458-13/default.htm#concept/c_xctu_layout.htm%3FTocPath%3DXCTU%2520overview%7C_____0. [Last entrance: 07 02 2019].

[29] «GNU General Public Licence,» GNU Operrating System, [En línea]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html. [Last entrance: 07 05 2019].

[30] «Diagram Circuits,» [En línea]. Available: https://www.circuit-diagram.org/editor/. [Last entrance: 24 04 2019].

[31] «Circuit Basics,» [En línea]. Available: http://www.circuitbasics.com/basics-uart-communication/. [Last entrance: 16 04 2019].

[32] «Circuit Basics,» [En línea]. Available: http://www.circuitbasics.com/basics-uart-communication/. [Last entrance: 12 05 2019].

[33] «Sparkfun,» [En línea]. Available: https://learn.sparkfun.com/tutorials/serial-communication/wiring-and-hardware. [Last entrance: 10 04 2019].

[34] «Sparkfun,» [En línea]. Available: https://learn.sparkfun.com/tutorials/serial-communication/wiring-and-hardware. [Last entrance: 12 05 2019].

[35] [En línea]. Available: https://learn.sparkfun.com/tutorials/serial-communication/all. [Last entrance: 23 05 2019].

[36] «Sparkfun,» [En línea]. Available: https://learn.sparkfun.com/tutorials/serial-communication/all. [Last entrance: 2019 02 01].

[37] «Luis Llamas,» [En línea]. Available: https://www.luisllamas.es/arduino-i2c/.

[Last entrance: 11 04 2019].

[38] [En línea]. Available: http://www.ieee802.org/15/pub/TG4.html.

[39] D. I. Inc., XBee®/XBee-PRO® RF Modules. Product Manual v1.xEx- 802.15.4 Protocol, 2009.

[40] «Stack Exchange,» [En línea]. Available: https://electronics.stackexchange.com/questions/37814/usart-uart-rs232-usb-spi-i2c-ttl-etc-what-are-all-of-these-and-how-do-th. [Last entrance: 16 04 2019].

[41] [En línea]. Available: https://www.techopedia.com/definition/19737/harvard-architecture. [Last entrance: 03 05 2019].

DOCUMENT 2. ANNEXES

CONTENT

ANNEX 1. CODE

These files can be found in the GitHub open repository named MPD in RamonHuesa account.

1 MPD for tests

This file is called MPDTest.ino and used in the MPD to read the values from the 3 accelerometers axis and 3 gyroscope axis from the MPU-6050 and send them through XBee transmission to the PC to be displayed and analysed in the steps designed to determine the algorithm to be used in the MPD.ino and MPD_M.ino versions of the MPD code.

GitHub:
https://github.com/RamonHuesa/MPD/blob/master/mpd_firmware/MPDTest.ino

```
//  <MPD Tests. Read values from the MPU-6050 and send them through the XBee.>
//    Copyright (C) <2019>  <Ramón Huesa Amat>
//
//    This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.
//
//    This program is distributed in the hope that it will be useful,
//    but WITHOUT ANY WARRANTY; without even the implied warranty of
//    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//    GNU General Public License for more details.
//
//    You should have received a copy of the GNU General Public License
//    along with this program; if not, see http://www.gnu.org/licenses
//    or write to the Free Software Foundation,Inc., 51 Franklin Street,
//    Fifth Floor, Boston, MA 02110-1301  USA

//    The acelerometre part based on MPU-6050 Short Example Sketch By Arduino User JohnChi
// The acelerometre part based on MPU-6050 Short Example Sketch By Arduino User JohnChi
#include <SoftwareSerial.h>     //library needed to use SoftwareSeral function
```

```
#include<Wire.h>      //library needed to use Wire funtion to configure MPU-6050
SoftwareSerial XBee(9,8); //XBee pins      //set pins 10 and 9 as serial comunication pins Tx and Rx
const int MPU_addr=0x68;      // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,GyX,GyY,GyZ;       //Inicialize ·variable for the Accelerometer axes and another 3 for the Gyro
void setup(){
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);     //Access PWR_MGMT_1 register
  Wire.write(0);      // set to zero desable sleep mode (wakes up the MPU-6050)
  Wire.endTransmission(true);
  XBee.begin(9600);     //9600 bps serial comunication with the XBee
}
void loop(){
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 registers
  AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L) it does not have to be read.
  GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
  XBee.print(" "); XBee.print(AcX);     //Transmiting the values to the serial communication with the XBee
  XBee.print(" "); XBee.print(AcY);
  XBee.print(" "); XBee.print(AcZ);
  XBee.print(" "); XBee.print(GyX);
  XBee.print(" "); XBee.print(GyY);
  XBee.print(" "); XBee.println(GyZ); XBee.print("\n");

  delay(10);      //Delay to help the communication to be read correctly
}
```

2 Matlab tests code

The following Matlab script has been used to read the data sent from the Arduino to the PC through XBee transmission. The data is read through the COM port where the USB XBee is plugged in.

This data is the “raw values” read from the MPU-6050 and which must have the same speed as the reading speed rate set before for the microcontroller-accelerometer and microcontroller-XBee communication (9600).

GitHub: https://github.com/RamonHuesa/MPD/blob/master/matlab/DisplayMatlabTest.m

```
%  <Matlab display. Display data received from the MPU via XBee-XBee USB.>
%    Copyright (C) <2019>  <Ramón Huesa Amat>
%
%    This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

%    This program is distributed in the hope that it will be useful,
%    but WITHOUT ANY WARRANTY; without even the implied warranty of
%    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%    GNU General Public License for more details.

%    You should have received a copy of the GNU General Public License
%    along with this program; if not, see http://www.gnu.org/licenses
%    or write to the Free Software Foundation,Inc., 51 Franklin Street,
%    Fifth Floor, Boston, MA 02110-1301  USA

clear all          % Clean the Workspace and Commands Window
delete(instrfind({'Port'},{'COM4'}));        % Delete some posible previous configuration for the port used for the XBee (COM4).

Rechazados=0;
extraDelay=35;
delay=10;          % Delay in the Arduino code
TestDuration=10;          % Duration of the test in seconds
SampleLength=round(TestDuration*1000/(delay+extraDelay));        % Samples=Duration[s]*(1000ms/1s)*(1sample/delay[ms])

tic          % Initiacilates tic funtion (used to track the time which every sample is taken at).
rate = 9600;          % 9600 bits per seconds which compared with the 50 seconds delay in the Arduino code is negligible
```

```matlab
s = serial('COM4', 'BaudRate',rate,'TimeOut',10,'Terminator','LF');  % Object created to represent the COM port, some inputs are set here
s.InputBufferSize = 1000000;

fopen(s);          % Open port communication

data=zeros(7,SampleLength);         % 7x100 matrix initialised with zeros which will be used to save every axis from the sensor (3 acceleration axis and 3 gyroscope axis) and the time vector.
i=1;
while i<=SampleLength         % When the SampleLength limit is reached the processor exits the loop and continues executing the rest of the code.
    sample=fscanf(s,'%i');
    if length(sample)==6;         % every reading block is checked to be completed with the 6 variables expected, as sometimes it is damaged and must be eliminated leaving those zeros behind.
        data(:,i)=vertcat(sample,toc);       % Variables are added to the matrix with the corresponding time extracted from “toc” function.
        i=i+1;
    else
        Rechazados=Rechazados+1;
    end
end
fclose(s);        % Close serial port bus

% Matrix divided into 7 arrays that are plotted to be visually analysed
Ax=data(1,:);
Ay=data(2,:);
Az=data(3,:);
Gx=data(4,:);
Gy=data(5,:);
Gz=data(6,:);
Time=data(7,:);

plot(Time,Ax,Time,Ay,Time,Az,Time,Gx,Time,Gy,Time,Gz);
legend('Ax','Ay','Az','Gx','Gy','Gz');
```

3 MPD Speed

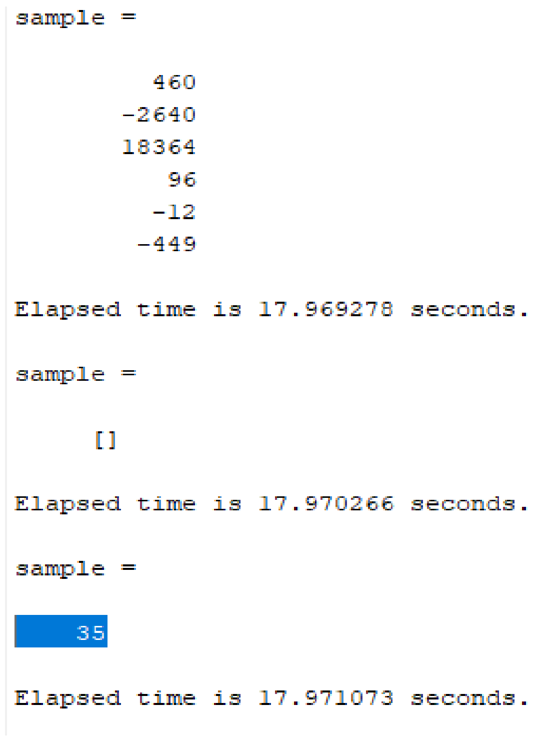
Millis() function offers the possibility to measure the time spent in a loop. If this time were close to cero the rate would correspond to a frame (with the 6 variables measured from the MPU) every 20ms corresponding to delay() function as the baud rate of 9600 is fast enough to be not considered.

Through the results with this test program we can assume the average duration of a void loop() cycle is 35ms. Thus every frame will be sent every 45 ms.

```
void loop(){
  int t=millis();
  Wire.beginTransmission(MPU_addr);     //ENTIENDO QUE ACCEDE A UN REGUISTRO LO GUARDA Y SE MUEVE
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 registers
  AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  //Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)   //yo este no lo voy a usar pero no se borrarlo y hacer que salte de Az a Gx
  GyX=Wire.read()<<16|Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)

  XBee.print(" "); XBee.print(AcX);     //Transmiting the values to the serial communication with the XBee
  XBee.print(" "); XBee.print(AcY);
  XBee.print(" "); XBee.print(AcZ);
  XBee.print(" "); XBee.print(GyX);
  XBee.print(" "); XBee.print(GyY);
  XBee.print(" "); XBee.println(GyZ);

  XBee.print("\n"); t=millis()-t;
  XBee.println(t);
delay(10);          //Delay to help the communication to be read correctly
}
```



```
sample =

         460
       -2640
       18364
          96
         -12
        -449

Elapsed time is 17.969278 seconds.

sample =

     []

Elapsed time is 17.970266 seconds.

sample =

    35

Elapsed time is 17.971073 seconds.
```

Figure 27. Matlab's output.

4 MPD_M

This file is called MPD_M.ino and the one used in the MPD to count steps and send the step count to the PC to be displayed with Matlab when the script is run. MPD evaluates the AcY values from the vertical accelerometer axis and add 1 step to the set count every time these values resemble a step vibration.

GitHub: https://github.com/RamonHuesa/MPD/blob/master/mpd_firmware/MPD_M.ino

```
//  <MPD_M. Counts steps from the MPU-6050 values and sends this count to the PC through XBee.>
//    Copyright (C) <2019>  <Ramón Huesa Amat>
//
//    This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.
//
//    This program is distributed in the hope that it will be useful,
//    but WITHOUT ANY WARRANTY; without even the implied warranty of
//    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//    GNU General Public License for more details.
//
//    You should have received a copy of the GNU General Public License
//    along with this program; if not, see http://www.gnu.org/licenses
//    or write to the Free Software Foundation,Inc., 51 Franklin Street,
//    Fifth Floor, Boston, MA 02110-1301  USA

//    The acelerometre part based on MPU-6050 Short Example Sketch By Arduino User JohnChi

#include <SoftwareSerial.h>      //library needed to use SoftwareSeral funtion
#include<Wire.h>      //library needed to use Wire funtion to configure MPU
SoftwareSerial xbee(9,8); //xbee pins      //set pins 10 and 9 as serial comunication pins Tx and Rx
const int MPU_addr=0x68;      // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,GyX,GyY,GyZ;      //Inicialize ·variable for the Accelerometer axes and another 3 for the Gyro
int stepCount=0;
int last_t=millis(), sPeed;
void setup(){
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);     //Access PWR_MGMT_1 register
  Wire.write(0);      // set to zero desable sleep mode (wakes up the MPU-6050)
  Wire.endTransmission(true);
  xbee.begin(9600);     //9600 bps serial comunication with the Xbee
```

```
}
void loop(){
  Wire.beginTransmission(MPU_addr);      //ENTIENDO QUE ACCEDE A UN REGUISTRO LO GUARDA Y SE MUEVE
  Wire.write(0x3D);  // starting with register 0x3D (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 registers
  AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)

  sPeed = millis()-last_t;
  if (AcY>18000)  {
        stepCount=stepCount+1;
        last_t=millis();

//Transmiting the values to the serial communication with the XBee
        xbee.print(" "); xbee.print(stepCount);
        xbee.print(" "); xbee.print(sPeed);
        xbee.print(" "); xbee.print(AcY);
        xbee.print("\n");
        delay(350);      //Delay to help the communication to be read correctly
  }
```

5 MPD

This file is called MPD.ino and the one used in the MPD to count steps and send the step count to the IUD to be displayed on the OLED display. MPD evaluates the AcY values from the vertical accelerometer axis and add 1 step to the set count every time these values resemble a step vibration.

GitHub:
https://github.com/RamonHuesa/MPD/blob/master/mpd_firmware/MPD.ino

```
//  <OLED display. Display data received from the MPU via Xbee.>
//    Copyright (C) <2019>  <Ramón Huesa Amat>
//
//    This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.
//
//    This program is distributed in the hope that it will be useful,
//    but WITHOUT ANY WARRANTY; without even the implied warranty of
//    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//    GNU General Public License for more details.
//
//    You should have received a copy of the GNU General Public License
//    along with this program; if not, see http://www.gnu.org/licenses
//    or write to the Free Software Foundation,Inc., 51 Franklin Street,
//    Fifth Floor, Boston, MA 02110-1301  USA

//    The accelerometre part based on MPU-6050 Short Example Sketch By Arduino User JohnChi

#include <SoftwareSerial.h>     //library needed to use SoftwareSeral funtion
#include<Wire.h>      //library needed to use Wire funtion to configure MPU
SoftwareSerial xbee(9,8); //xbee pins      //set pins 10 and 9 as serial comunication pins Tx and Rx
const int MPU_addr=0x68;      // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,GyX,GyY,GyZ;      //Inicialize ·variable for the Accelerometer axes and another 3 for the Gyro
int stepCount=0;
int lapse=millis();
void setup(){
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);     //Access PWR_MGMT_1 register
  Wire.write(0);      // set to zero desable sleep mode (wakes up the MPU-6050)
  Wire.endTransmission(true);
  xbee.begin(9600);     //9600 bps serial comunication with the Xbee
}
```

```
void loop(){
  Wire.beginTransmission(MPU_addr);      //ENTIENDO QUE ACCEDE A UN REGUISTRO LO GUARDA Y SE MUEVE
  Wire.write(0x3D);  // starting with register 0x3D (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 registers
  AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)

  if ((AcY>18000)&&((millis()-lapse)>0.35))  {
        stepCount=stepCount+1;
        lapse=millis();

//Transmiting the values to the serial communication with the Xbee
        xbee.print(AcY);
        xbee.print(lapse);
  }

delay(10);      //Delay to help the communication to be read correctly
}
```

6 Matlab Display

This file is called DisplayMatlab.m and is the one used in the PC as a Matlab script to display the step count received from the MPD. The transmitting MPD must be using MPD_M.ino.

Every time the PC receive a package of data, including the step count, from the MPD saves it in a matrix and shows the step count in the command window.

This script keeps running even if the MPD is turned off. It ends the loop when the maximum steps are reached or when a intense vertical acceleration is received, it can be rather a jump, a strong step or a soft hit on the MPD with the fingertip.

GitHub:
https://github.com/RamonHuesa/MPD/blob/master/matlab/DisplayMatlab.m

```
%  <Matlab display. Display data received from the MPU via XBee-XBee USB.>
%    Copyright (C) <2019>  <Ramón Huesa Amat>
%
%    This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.

%    This program is distributed in the hope that it will be useful,
%    but WITHOUT ANY WARRANTY; without even the implied warranty of
%    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%    GNU General Public License for more details.

%    You should have received a copy of the GNU General Public License
%    along with this program; if not, see http://www.gnu.org/licenses
%    or write to the Free Software Foundation,Inc., 51 Franklin Street,
%    Fifth Floor, Boston, MA 02110-1301  USA

%clear all          % Clean the Workspace and Commands Window
delete(instrfind({'Port'},{'COM4'}));        % Delete some posible previous configuration for the port used for the Xbee (COM4).

TestDuration=30;         % Duration of the test in seconds
Rechazados=0;
extraDelay=35;
delay=10;          % Delay in the Arduino code
SampleLength=200;       % Samples=Duration[s]*(1000ms/1s)*(1sample/delay[ms])

```

```
tic          % Initiacilates tic funtion (used to track the time which every sample is taken at).
rate = 9600;          % 9600 bits per seconds which compared with the 50 seconds delay in the Arduino code is negligible
s = serial('COM4', 'BaudRate',rate,'TimeOut',10,'Terminator','LF');        % Object created to represent the COM port, some inputs are set here
s.InputBufferSize = 1000000;

fopen(s);          % Open port communication

data=zeros(3,SampleLength);          % 7x100 matrix initialised with zeros which will be used to save every axis from the sensor (3 acceleration axis and 3 gyroscope axis) and the time vector.
i=1;
AcY = 0;
last_t=0;
stepCount=0;
while ((i<=SampleLength)&&(AcY<30000))          % When the SampleLength limit is reached the processor exits the loop and continues executing the rest of the code.
    sample=fscanf(s,'%i');
    if length(sample)==3;          % every reading block is checked to be completed with the 6 variables expected, as sometimes it is damaged and must be eliminated leaving those zeros behind.
        data(:,i)=vertcat(sample);        % Variables are added to the matrix with the corresponding time extracted from “toc” function.
        stepCount=data(1,i)
        pSpeed=data(2,i);
        AcY = data(3,i);
        i=i+1;

    else
        Rechazados=Rechazados+1;
    end

end
fclose(s);          % Close serial port bus

s="End of the test";
disp(s)
```

7 UID

This fille is called IUD.ino and is the one used in the UID to display on the OLED display the step count received from the MPD. MPD must be using the MPD.ino fille.

This code has not been tested with real values in an UID so some adjustments might be needed.

GitHub: https://github.com/RamonHuesa/MPD/blob/master/UID.ino

```
//  <OLED display. Display data received from the MPU via Xbee.>
//    Copyright (C) <2019>  <Ramón Huesa Amat>
//
//    This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or any later version.
//
//    This program is distributed in the hope that it will be useful,
//    but WITHOUT ANY WARRANTY; without even the implied warranty of
//    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//    GNU General Public License for more details.
//
//    You should have received a copy of the GNU General Public License
//    along with this program; if not, see http://www.gnu.org/licenses
//    or write to the Free Software Foundation,Inc., 51 Franklin Street,
//    Fifth Floor, Boston, MA 02110-1301  USA


#include <Wire.h>       // libraries needed to use OLED
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>

#include <SoftwareSerial.h>     // library needed to use Xbee

#define SCREEN_WIDTH 128    // OLED display width, in pixels
#define SCREEN_HEIGHT 64    // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_ADDR   0x3D     // Address 0x3D for 128x64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

 SoftwareSerial xbee(10,9);      // set pins 10 and 9 as serial comunication pins Tx and Rx

  int iniMPDTime, sesionAverage, stoppedTime,currentMillis, MPDTime;
  int stepCount = 0;
  int t = 0;

void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR); // initialize and clear display
  display.clearDisplay();
  display.display();

  xbee.begin(9600);      //9600 bps serial comunication with the Xbee

// display inicializatin text
  display.setTextSize(1);     // Normal 1:1 pixel scale
  display.setTextColor(WHITE);     // Draw white text
  display.setCursor(0,20);     // Start at 20 pixels from the top and left
  display.print("Welcome to MPD prototype");
  display.setCursor(0,40);         // Start at 40 pixels from the top and left
  display.print("by Ramon Huesa Amat");

  display.display();     // update display with all of the above graphics
}

void loop() {

stepCount = xbee.read();
t = xbee.read();

if ((stepCount==0)&&(t==0)) {
iniMPDTime=t;
}

  // display a line of text
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0,20);
  display.print("Step Count");
  display.setCursor(0,40);
  display.print(stepCount);


  // update display with all of the above graphics
  display.display();

// Extra usefull data for the user (not displayed)
currentMillis=millis();
MPDTime = currentMillis+iniMPDTime;
sesionAverage = 1000*stepCount/currentMillis;
stoppedTime= MPDTime-t;

}
```

8 License

The code made for this project has been fed by other free software codes shared online For that reason and in order to contribute to educative purposes, all the code used for this project use **GNU General Public License** (**GNU GPL** or **GPL**), which guarantees end users the freedom to run, study, share and modify the software. The GPL is a copyleft license, which means that derivative work can only be distributed under the same license terms [29].

GNU version 3 is the current latest version. The foundation of this license is that there are four freedoms that every user should have:

- the freedom to use the software for any purpose
- the freedom to change the software to suit user's needs
- the freedom to share the software
- the freedom to share the changes you make

An important condition of this license is that developers who write software can release it under the terms of the GNU GPL, this makes the software irreversibly become free software and remain free software, no matter who changes or distributes the program. That is what copyleft means: in the same way software is copyrighted, copyleft uses those rights not to restrict users like proprietary software does, but so they use them to ensure that every user has this freedom.

The license declaration is commented at the beginning of each one of the source code files used in this project.

ANNEX 2.
ASSEMBLY AND DRAWINGS

1 MPD

The MPD is the step counting unit and the most important part of this project.

1.1 Assembling

The current MPD version is the third one. The previous two were discarded and its design improved. It follows a breve explanation:

<u>MPD v1</u>

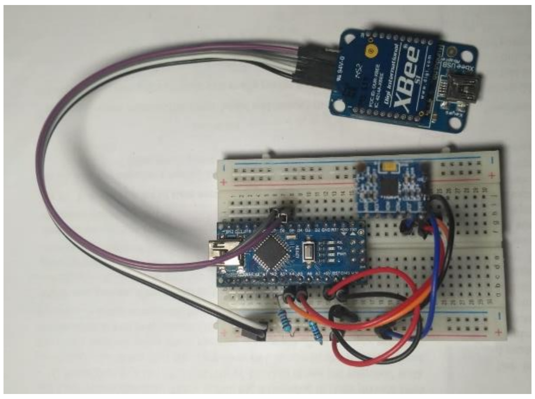


Figure 28. MPD v1.

MPD v1 (Figure 28) was built on bread board and was easy to adjust connections but the difficulty to maintain the cable connections while it was coupled to the user made changing into a metal board necessary.

<u>MPD v2</u>

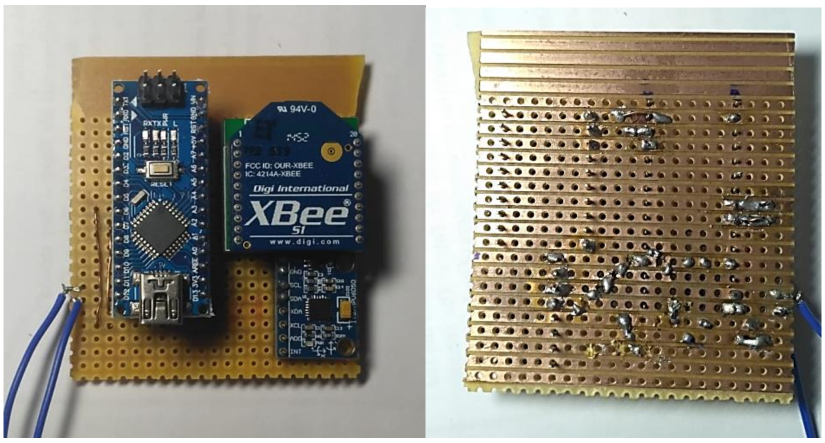


Figure 29. MPD v2 both sides.

The soldering in the MPD v2 (Figure 29) where inconsistent and the protecting shield which it was dimensioned for was too large to be brought comfortably. Another problem with this model was that the charging module had been added after and did not have enough free space on the board.

<u>MPD v3</u>

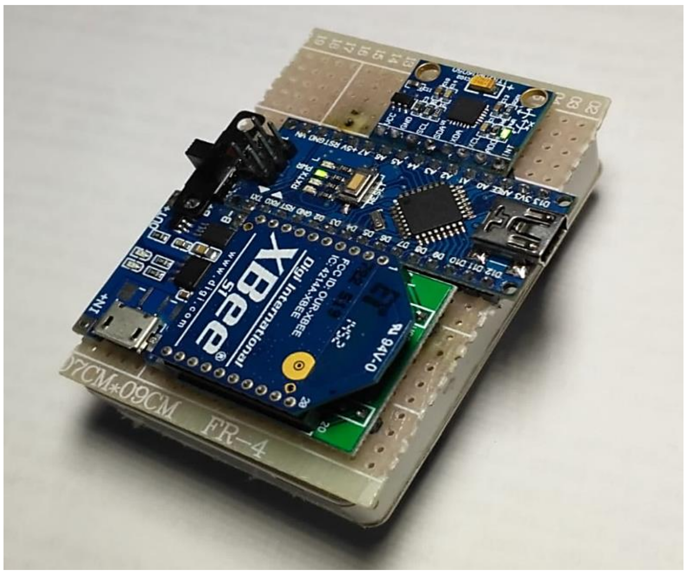


Figure 30. MPD v3.

MPD v3 (Figure 30) solved the previous problems and improved further. It is far sturdier, more precisely soldered, and has a more appealing design.

Fort he MPD the following components are required:

Plastic Buckle Webbing Ending Clip: Figure 31. Cutting one side standing edges and sanding the surface will make it ready to be glued to the back side of the shield. This component is intrusted with the task of coupling the device to the top border of the user's trousers.

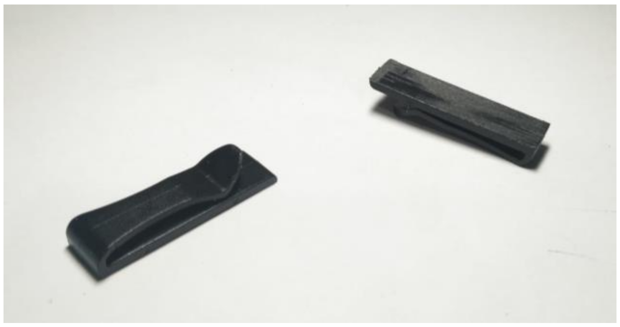


Figure 31. Buckle Webbing Ending Clip cut and sanded.

Plastic Shell: Figure 32. As the original box is unnecessarily high it has been sliced in half using the solderer. Will be kept only the top part and gluing a second cover on its bottom part where the clips will be fixed.

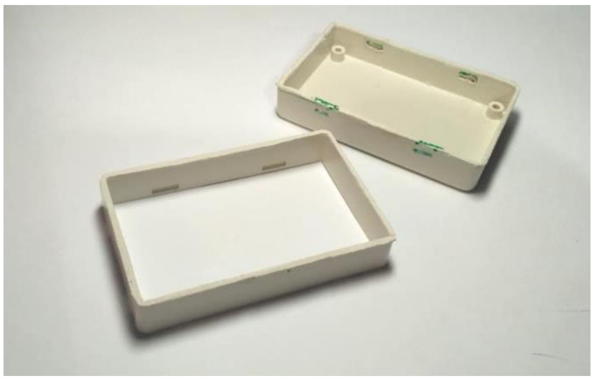


Figure 32. Case Enclosure Box cut in half.

Figure 33 shows the resulting structure:

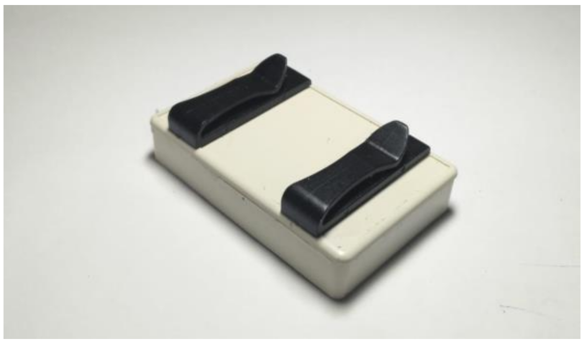


Figure 33. Clip + Cover + Half-Box assembling.

The cover is then placed on the other side with the battery glued to it in order to minimise the risk of being hit (Figure 34).

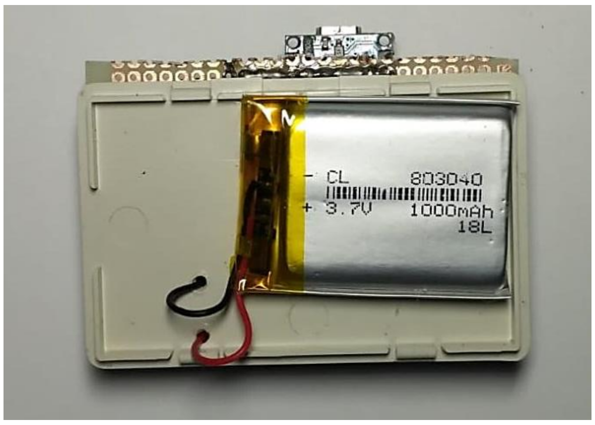


Figure 34. Lithium battery glued to the shield cover.

Its cables pass through two circular holes made with the solderer tip. On top of this cover the circuit board will be fix glued with 2 pieces of sponge material in anticipation of possible modifications that requires to separate them by easily breaking the sponges and this way access to the solderings (Figure 35).

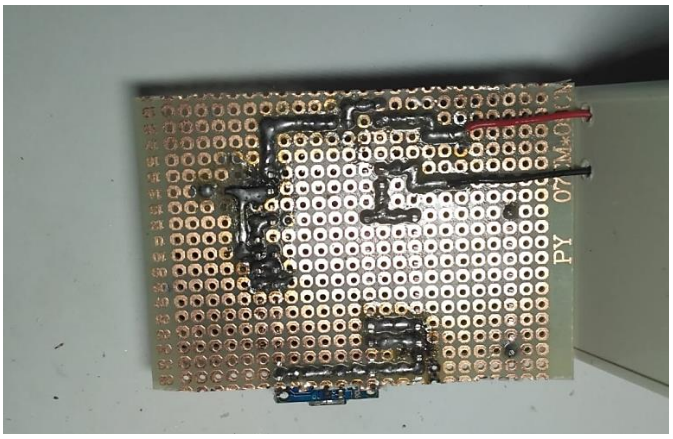


Figure 35. Soldering on the board back.

1.2 Electronic Schematic

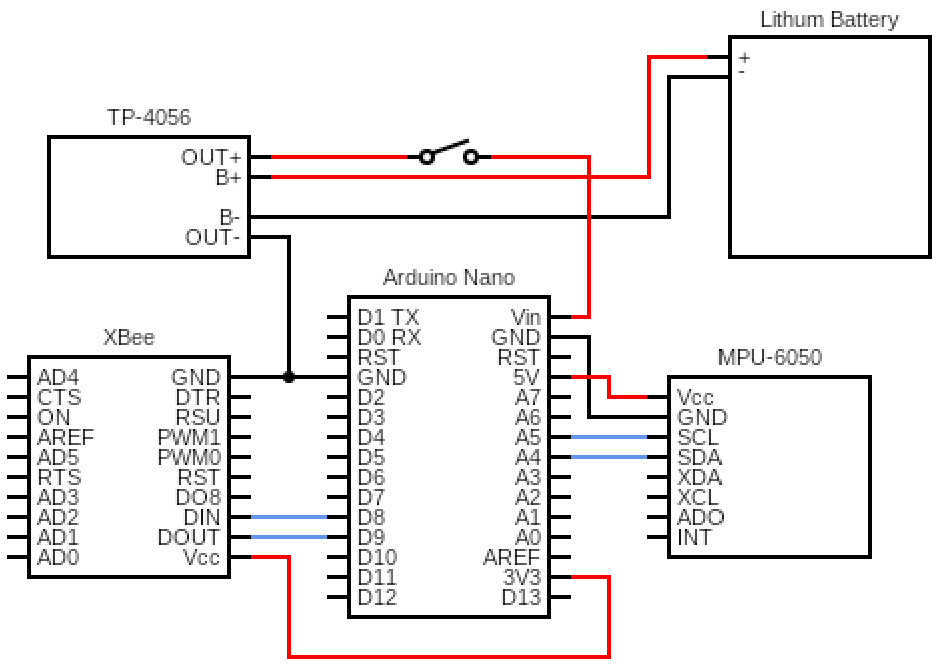


Figure 36. MPD connection diagram created with Diagram Circuits [25]

In Figure 36 diagram wires are classified by colours:

- Black: Ground, 0V.
- Red: Direct voltage, 5/3.3V.
- Blue: Communication.

Arduino-MPU-6050serial communication follows I2C protocol. In the Arduino code, Wires function is set to communicate with the MPU, Arduino Nano pins for I2C communication are fixed to pins A5 (SDA) and A5 (SCL).

Arduino-XBee serial communication follows instead TTL Serial protocol. In the Arduino code, SoftwareSerial function is set to communicate with the XBee but in this case pins used for this communication can be specified in the code among various digital pins, for this assembly D9 (Rx) and D8 (Tx) are chosen for its physical proximity with The XBee module pins.

In terms of power supply, the battery supplies 3.85V which the Arduino internally converts into rather 5V or 3V3. XBee is powered with the 3V3 pins whereas the MPU-6050uses 5V output pin.

2 UID

2.1 Assembling

Due to a broken XBee module the hardware assembling of the UID unit has been impossible in the time and assigned budget for this project. Nevertheless, all the documentation of its components, schematics and code are contained in this paper to allow its construction or design modifications. Figure 37 shows the distribution of the components on the circuit board:

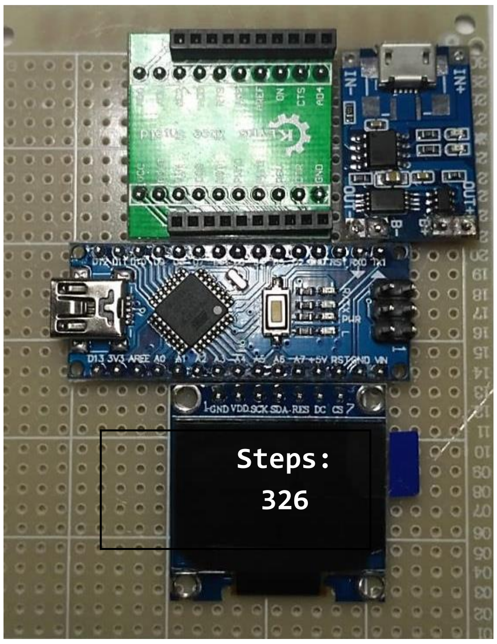


Figure 37. IUD board.

In absence of the UID, the output display of the step count from the MPD can be obtained from the PC by using the Matlab script.

2.2 Electronic Schematic

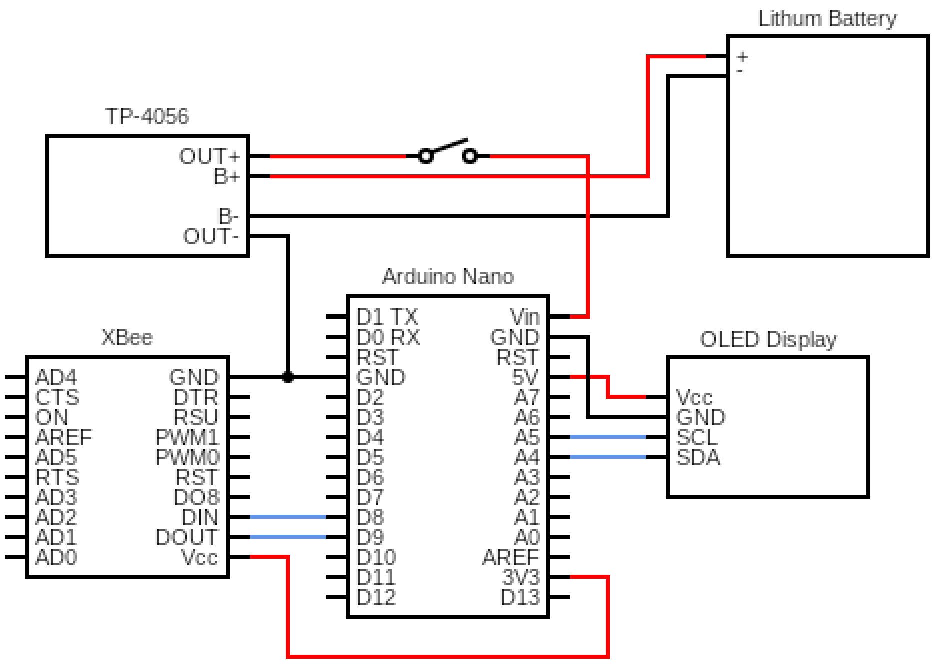


Figure 38. UID connection diagram created with Diagram Circuits [25]

In Figure 38 diagram wires are classified by colours in the same way as in the previous MPD:

- Black: Ground, 0V.
- Red: Direct voltage, 5/3.3V.
- Blue: Communication.

Arduino-OLED display communication follows I2C protocol whereas Arduino-XBee communication follows TTL Serial protocol. Both protocols explained in the MPD assembling and in more detail in the Annex 3. Communication Protocols.

3 Budget

The total money expended in this project can be calculated by summing all the component prices (delivery is included) and the value of the material used in the assembling multiplied by an estimated percentage of it used.

ITEM	Price/Unit	Units	Price
Arduino Nano	1.92€	2	3.84 €
MPU-6050	1.60€	1	1.60 €
XBee s1		2	0.00 €
OLED Display	1.52€	1	1.52 €
XBee Adaptor	1.19€	2	2.38 €
TF4056	0.41€	2	0.82 €
Lithium Battery	2.20€	2	4.40 €
Circuit Board	0.61€	2	1.22 €
Shield	0.57€	4	2.28 €
Clips	0.61€	4	2.44 €
Glue	0.77€	0.4	0.31 €
TOTAL			20.81€

Table 7. Materials budget.

All the components where bought from Aliexpress.com and Amazon.es except the XBee that are not included in the budget as they are borrowed from Universidad de Almería. Another project should consider the cost of them.

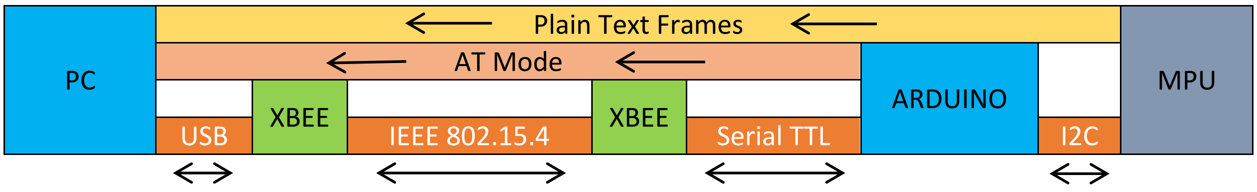
Apart from this price, must be considered the use of other tools needed that had to be bought for this project: electronic solderer, tin wire, flux and material used for previous prototypes.

It must also be considered that a cost coming from the manufacturing exist that can hardly be estimated from the assembling of these prototypes such as efficiency in this assembling would be reached through repetition and the method would be improved. On the other hand, assigning a cost for a prototype mass production would have a real interest.

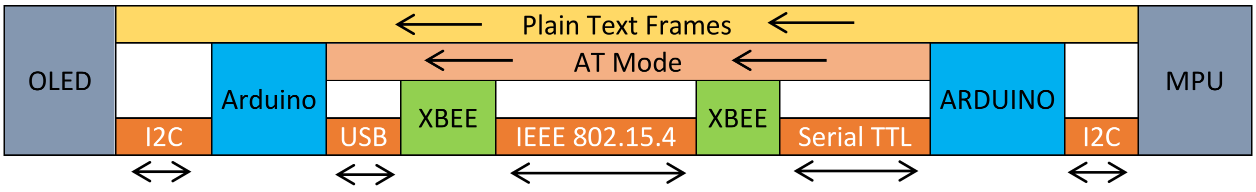
ANNEX 3. COMMUNICATION PROTOCOLS

1 Communication Schematic

1.1 PC displayed



1.2 OLED displayed



These graphs illustrate the data flow between all the components as well as all the communication protocols involved in the process.

There are two possible communication network designs, the first one is using the PC as end of the net and the other one is using the OLED display instead.

In the base of the communication layers are the basic protocols of communication used by each component: I2C, IEEE 802.15.4, Serial TTL and USB.

The second layer represents the AT mode in the XBee communication that allows the two components to communicate in a similar way as they were connected directly to each other. This mode is also called Transparent Mode.

The top layer represents the communication protocol created with the combination of all the protocols in the nets and consists of plain text frames sent by the MPU rather than the PC display or the OLED display. This protocol is one-way communication coming from the MPU-6050 to the PC or the OLED display.

2 Serial TTL communication

Serial communication is the process of sending data one bit at a time, sequentially, over a communication channel. Parallel communication instead, is created by several busses transmitting several bits at a time. TTL (Transistor Transistor Logic) is an old technology for digital logic and the name is often used to refer to the 5 V supply voltage.

2.1 UART

UART stands for Universal Asynchronous Receiver/Transmitter. It is not a communication protocol like SPI and I2C, but a peripheral in the microcontroller. The electric signalling levels and methods are handled by a driver circuit external to the UART, if not using TTL/CMOS logic levels.

2.2 Wiring and Hardware

In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from controlling device into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs, one for sending data and another for receiving. As such, serial devices should have two serial pins: the receiver, **RX**, and the transmitter, **TX**. It's important to note that those *RX* and *TX* labels are with respect to the device itself. So data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART, and vice-versa (Figure 39) [28] .

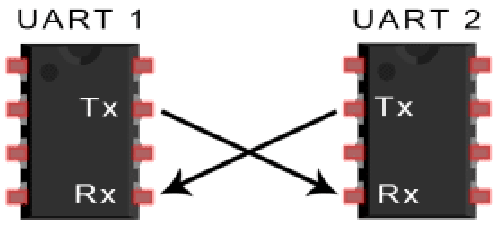


Figure 39. UART wire connection [30].

For some serial busses just a single connection between a sending and receiving device might be enough if one of them does not need to send any data back, which makes the communication unidirectional.

When two serial devices are connected together, it is important to verify their signal voltages match up.

2.2.1 Protocol

The UART that is going to transmit data receives the data from a data bus. The data bus is used to send data to the UART by another device like a CPU, memory, or microcontroller. Data is transferred from the data bus to the transmitting UART in parallel form. After the transmitting UART gets the parallel data from the data bus, it adds a start bit, a parity bit, and a stop bit, creating the data packet. Next, the data packet is output serially, bit by bit at the Tx pin. The receiving UART reads the data packet bit by bit at its Rx pin. The receiving UART then converts the data back into parallel form and removes the start bit, parity bit, and stop bits. Finally, the receiving UART transfers the data packet in parallel to the data bus on the receiving end [29].

A serial interface where both devices may send and receive data is either **full-duplex** or **half-duplex**. Full-duplex means both devices can send and receive simultaneously. Half-duplex communication means serial devices must take turns sending and receiving.

2.3 Asynchronous

UARTs transmit data *asynchronously*, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred (Figure 40). Both UARTs must be configured to transmit and receive the same data packet structure.

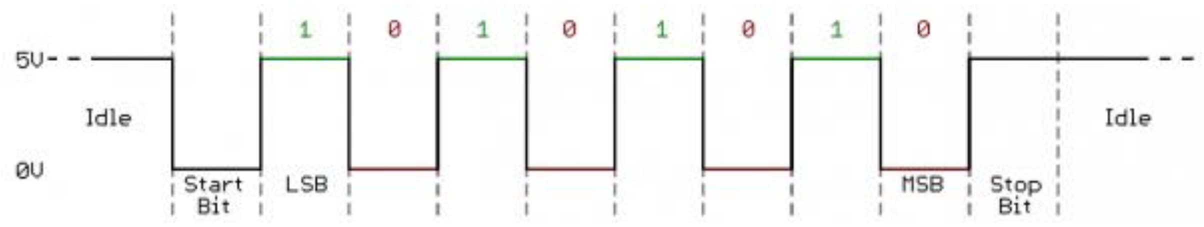


Figure 40. Serial frame signal [31].

The Arduino TTL Serial is an asynchronous serial protocol which has a number of built-in rules - mechanisms that help ensure robust and error-free data transfers. These mechanisms, which we get for eschewing the external clock signal, are:

- Baud rate
- Synchronization bits
- Parity bits

2.3.1 Baud rate

This protocol is highly configurable but it is important to make sure that both connected devices are configured to use the exact same protocol.

The baud rate specifies data speed sent over a serial line. It is usually expressed in units of bits-per-second (bps). This value determines how long the transmitter holds a serial line high/low or at what period the receiving device samples its line.

One of the more common baud rates, especially for simple stuff where speed is not critical, is 9600 bps. This baud rate will be used in many Arduino code examples.

2.3.2 Data framing

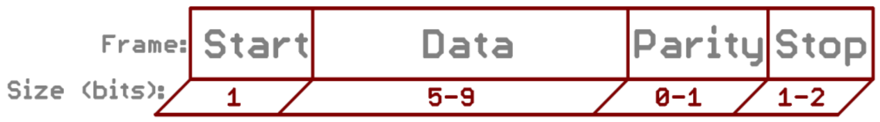


Figure 41. Serial Frame structure [35].

Each data transmitted is sent in packet or frame of bits, as in Figure 41. Frames are created by appending synchronization and parity bits to our data.

The real information carried in a frame is the data piece. This block of data is also called *chunk* because its data is not specifically stated. The size of the data piece is something in between of 5-9 bits; data size will usually be 8 bits as the standard (1 byte).

2.3.3 Synchronization bits

The synchronization bits are 1 at the beginning and 1 o 2 at the end, and as supposed, they mark the beginning and the end of the frame. It tells the receiver when data is starting and when it is finished.

2.3.4 Parity bit

The parity bit is used just as a confirmation of the data received that it was correctly read. It works checking if the data is an odd or an even number showing 1 or 0 in each case. Assuming that if there is any error it will be in just one bit, otherwise it would work with even number of error. Parity is optional and not very widely used but it can be helpful to protect communication against noise. [26]

3 I2C Communication

Along with serial port and SPI bus, I2C (Inter-Integrated Circuit) is one of the 3 main communication systems in Arduino environment [30].

3.1 Wiring and Hardware

I2C, also known as TWI (Two Wired Interface), just requires two wires for full communication, one used for the clock signal (CLK) and the other one for sending and receiving data.

In the bus every device has an address number assigned to access to every single one individually. This address can rather be assigned in the hardware, which usually give the programmer the possibility of changing the last 3 bits through jumpers or interrupts, or completely by the code.

Every single device connected to the bus must have a unique address number, otherwise it must be changed or, in case that this is impossible, enable another bus.

3.2 Protocol

I2C protocol is based in master-slave architecture. The master device begin communication with slaves and then slaves can reply to the master. Slaves cannot begin a conversation if the master does ask them first or either talk directly to each other.

There is normally just one master, despite multiple masters buses being possible, just one of them can be the master at a time and master changes introduce high complexity.

3.3 Synchronous

The I2C protocol is synchronous. The master provides a clock signal that keeps all the devices in the bus synchronised. Thus, there is no need for every single device to have its own clock signal, to arrange a transmission speed and to have mechanisms to maintain transmission synchronised as in UART protocol.

I2C protocol considers Pull-Up resistors from wires to Vcc as shown Figure 42. In Arduino projects, seeing it not installed is frequent as Arduino library *Wire* activates internal Pull-Up resistors. Withal, internal resistors have an impedance in the range of 20-30 KOhm, so these are very strong resistors.

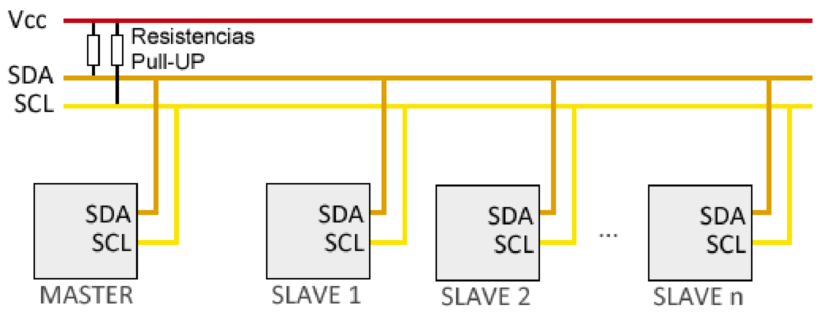


Figure 42. I2C protocol wiring [30].

This makes signal rises slower and it involve lower baud rates and shorter communication distances. To avoid these problems extra Pull- Up resistors must be added to the circuit of a value in between 1 and 4.7 KOhm.

3.4 Data framing

In order to perform good communication with just a single wire, I2C bus uses large frames, as shown in Figure 43, conformed by these elements:

- 7 bits: address number of the device to communicate to.
- 1 bit: to indicate if the request is to send data or receive it.
- 1 bit: validation.
- 1 or more bits: data sent or receive from the slave.
- 1 bit: validation.

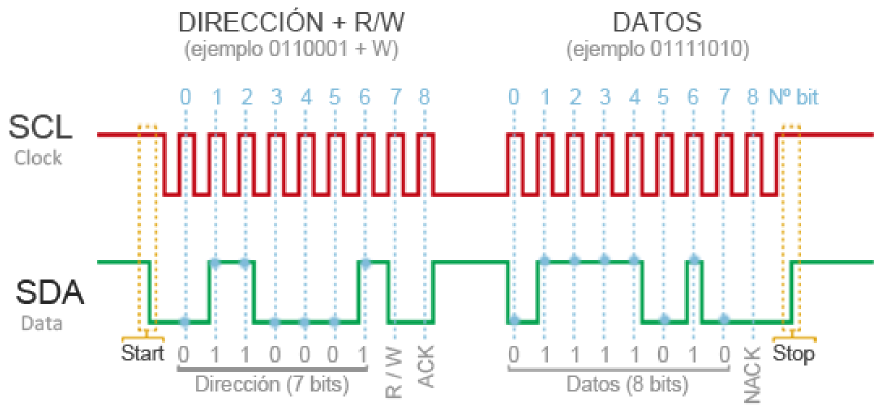


Figure 43. I2C frames signal.

With these 7 bits it is possible to access to 112 different devices in a single bus (16 address numbers are set for special utilities).

If the data piece is 8 bits (1 byte) the data frame grows up to 18 bits, which makes the communication not as fast as in other protocols. Standard transmission rate is 100MHz, and high speed mode around 400MHz.

I2C standards consider other working modes with longer address numbers or slower speeds but they are not usually implemented in Arduino.

One of the perks of using I2C communication is the verification of successful transmission, although there is no verification of correct content of the message and the communication cannot be full-duplex.

4 IEEE 802.15.4

XBee modules support multiple wireless protocols which are suitable for different network topologies.

IEEE 802.15.4 is a technical standard which defines the operation of LR-WPANs (low-rate wireless personal area networks). It specifies the physical layer and media access control for LR-WPANs. XBee further extends the standard by developing the upper layers which are not defined in IEEE 802.15.4 [32].

5 AT Mode

By default, XBee Modules operate in Transparent Mode, however it is recommended to directly configure this mode following the instruction in the XCTU section. When operating in this mode, the modules act as a serial line replacement - all UART data received through the DI pin in the XBee is queued up for RF transmission. When RF data is received, the data is sent through pin DO in the Xbee [33].

Data is buffered in the DI buffer until one of the following causes the data to be packetized and transmitted:

- No serial characters are received for the amount of time determined by the RO (Packetiza-tion Timeout).
- The maximum number of characters that will fit in an RF packet is received (100 characters)
- The Command Mode Sequence (GT + CC + GT) is received.

If the module cannot immediately transmit (for instance, if it is already receiving RF data), the serial data is stored in the DI Buffer until it can transmit it.

If the DI buffer becomes full, hardware or software flow control must be implemented in order to prevent overflow (loss of data between the host and module).

ANNEX 4. ALGORITHM DEVELOPMENT

Algorithm method

Analysing the plot after some test it seems the axis Ay (the vertical accelerometer axis) and Gz (the horizontal sided gyroscope axis) shows a higher response to steps. The reason for that is when a step is made there is a vertical hit with the floor that generates vibration and this is picked up by the Accelerometer specially in the Y axis, likewise shortly before the step the initiation of the movement is made by a hip flexion which is received in the gyroscope as a rotation in the Z axis.

This algorithm used to detect steps just used the variable Ay, as it is enough to perceive the step movements. When we look at the Ay plot using the command >>plot(Ay), if the test has been done previously saving the variables, we can appreciate the steps as high oscillations on the graph. There are two main conditions to implement the algorithm.

Condition 1. The first one is to define what is the trigger value in order to assess a raise in Ay as a peak created by a step. If the trigger value is too low other body movements can be considered steps by the MPD and it could add these false steps to the step count. On the other hand, if this value is too high then soft steps in slow walking will not be detected.

Condition 2. The other condition, and this one is a bit more sensitive, is to bear in mind that due to imprecisions in the measurements the signal may have picks without a step movement or a low pick while a step pick is occurring without a clear movement reason. Because of this, two separated picks in a single step movement can happen.

This can be solved with a “wait” condition in the algorithm by not adding new steps to the count until a certain time has passed. The big problem here is that if this time is too short there might be double steps separated enough to be saved in the step count. But if it is too long the MPD would lose its ability to measure fast walking.

These two values have been adjusted by tests with the algorithm implemented in Matlab with an if() condition where the real steps numbers where counted aloud while the user was performing the test and at the same time checking real-time the correspondence on the PC.

Tests

Here is an explanation of how the tests to develop the algorithm were performed.

In the test 3 kind of waking are consider:

Fast walking: Not completely maximum speed but still fast and difficult to maintain for a long time. Speed walking of people walking in the street with a destination and not much time to arrive there. Up to 120 steps per minute.

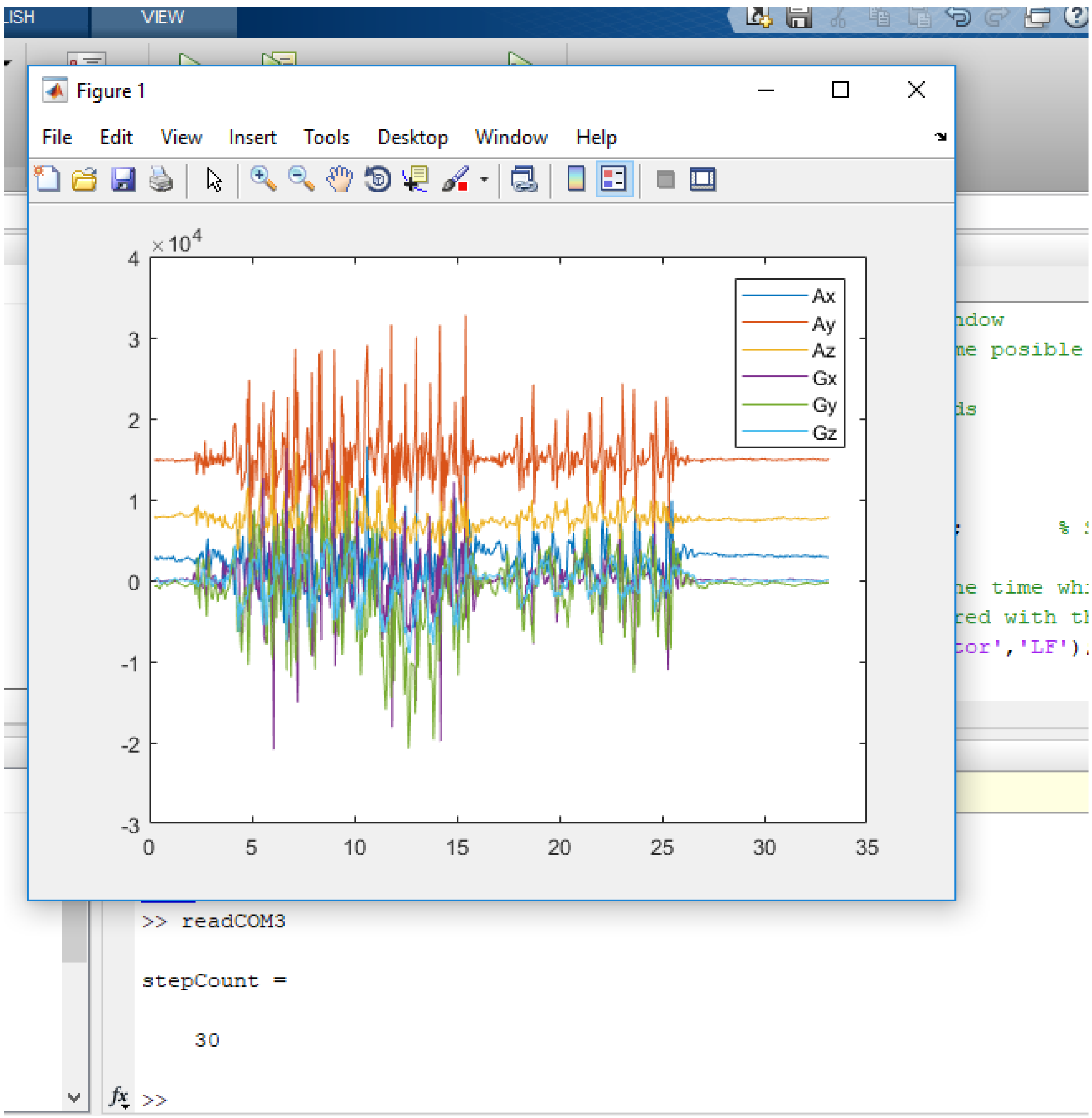
Medium speed waking: Average walk for people. Approximately 85 steps per minute.

Slow walking: Short steps with variable speed in the way of a person moving inside of a room.

Short displacements with very short steps are not considered in the algorithm, the MPD may detect some of them but because of “Condition 2” fast short waking will not add many steps in a short time. This algorithm is just orientated to measure natural continuous walking.

The test is done with a tester and a tested subject, tests were done in 3 different people train to minimise the effect of personal differences in the efficiency of the MPD and this way make the device suitable for as many kind of users as possible.

This test was performed in order to adjust the 2 parameters in the algorithm and find a correspondence between the program output and reality.



```
>> readCOM3

stepCount =

    30

>>
```

Figure 44. Test with the MPU.

Figure 44 shows the result seen after a test is performed the stepCount variable count steps and in the test the result was exactly the same as in the real model. By just plotting the Ay a better analysis can be done.

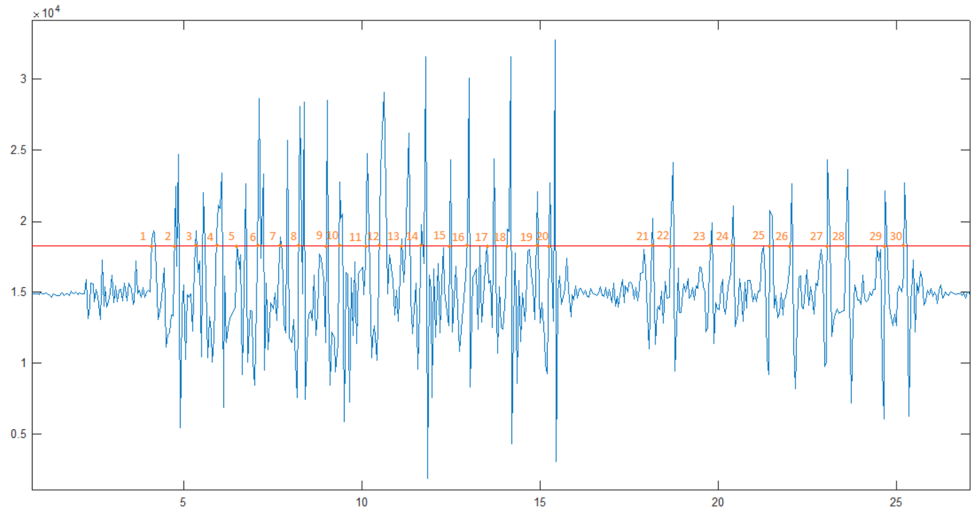


Figure 45. Matlab plot of Ay for the MPD tests.

In Figure 45, the plot of one of the test can be seen. This test consisted in 20 medium-speed steps and 10 slow-speed steps and the algorithm successfully assigned all the steps.

The test is performed using the code MPDtest.ino for the Arduino and DisplayMatlabTests.m for Matlab. The commented code can be seen in the Annex 1. Code.

Resumen/Abstract

El sedentarismo es una de las causas principales de muerte prevenible y morbilidad en los países desarrollados. Los actuales niveles de inactividad física son en parte debidos a una participación insuficiente en actividades físicas y un aumento de comportamiento sedentario durante las actividades diarias. Caminar más es una solución adecuada para la mayoría de las personas, y midiendo la cantidad de actividad al caminar, los usuarios dispondrán de una herramienta útil para aumentar su actividad física.

Este proyecto aborda el reto de detectar pasos mediante el uso del acelerómetro, controlado por microcontrolador Arduino y transmitirlo inalámbricamente a otro terminal para ser visualizado. Este diseño corresponde al prototipo destinado a ayudar en el desarrollo de una versión de podómetro posterior que pueda ser fabricada con un diseño más compacto y con funcionalidades adicionales.

Physical inactivity is a leading cause of preventable death and morbidity in developed countries. The current levels of physical inactivity are partly due to insufficient participation in physical activities and an increase in sedentary behaviour during daily activities. Increase walking is a suitable solution for most people, and by measuring the amount of activity in the walk, people will have a useful tool to increase their physical activity.

This project addresses the challenge of detecting steps through the use of the accelerometer, controlled by the Arduino microcontroller and transmit it wirelessly to another terminal in order to be displayed. This design is a prototype intended to support the development of a later pedometer version that can be manufactured with a more compact design and further functionalities.