STUDY AND IMPLEMENTATION OF A FALL DETECTION SYSTEM BY MEANS OF AN ACCELEROMETER

MASTER THESIS

Student	**Lucia Gallego Olivares**
Date	Academical year 2018/2019
Director Thesis	Niccolò Mora
Course	2018/2019

ABSTRACT

Fall detection is an important area of research due to its close relation with healthcare, especially for the elderly people. There is a need to design a reliable and quick system to minimise the possible injuries once the fall has occurred, related to providing the fastest intervention possible by emergency services. The aim of this work is to develop a reliable system that is capable to obtain the acceleration signal when the subject is doing normally daily activity and to identify fall detection in real-time. First, it will be determined a suitable position of the sensor for the acquisition of the acceleration data during the fall. Secondly, a hardware system will be developed for the acquisition of data based on accelerometers. The hardware system consists on an accelerometer and a microcontroller, powered by an external battery. The microcontroller is the responsible of reading the output signal of the accelerometer by an SPI port, and send it to the PC trough WIFI. Likewise, it will be developed the MATLAB program that allows reading and analysing data for falling detection. For that, it is proposed in this thesis determine the characteristical parameters of the acceleration signal that allow to make the difference between normal steps and falls.

Key Words: Fall detection, accelerometer, elderly.

RESUMEN

La detección de caídas es un área importante de investigación debido a su estrecha relación con la atención médica, especialmente para las personas mayores. Es necesario diseñar un sistema fiable y rápido para minimizar las posibles lesiones una vez que se ha producido la caída, de manera que se ejecute la intervención por parte de los servicios de emergencia lo más rápido posible. El objetivo de este trabajo es desarrollar un sistema fiable que sea capaz de obtener la señal de aceleración cuando el sujeto está llevando a cabo una actividad diaria y ser capaz de identificar la detección de caídas en tiempo real. En primer lugar, se determinará una posición adecuada del sensor para la adquisición de la señal de aceleración durante la caída. A continuación, se desarrollará el sistema de hardware para la adquisición de datos basados en acelerómetros. El sistema de hardware se compone de un acelerómetro y un microcontrolador alimentado por una batería externa. El microcontrolador es el responsable de leer la señal de salida del acelerómetro por un puerto SPI, y enviarlo al PC a través de una red WIFI. Asimismo, se desarrollará el programa de MATLAB que permita la lectura y análisis de los datos para la detección de caídas. Para ello, se propone en esta tesis determinar los parámetros característicos de la señal de aceleración que permiten hacer la diferencia entre pasos normales del sujeto y caídas.

Palabras: Detección de caídas, acelerómetro, personas mayores

INDEX

INDEX OF PICTURES

INDEX OF ILLUSTRATIONS

INDEX OF TABLES

1 INTRODUCTION

The constant increase in the number of networked devices and the increased interest in the sector of Internet of things (IoT) by companies, universities and private individuals are the starting point for this academic project, which sets as its main objective the realization of a smart system based on a simple smart object. The fields of application of these intelligent devices are multiple depending on the sector and purposes. The smart object chosen for this project is to develop a fall detection system.

Elderly people are known to be more prone to unexpected dangerous falls, only because they are more fragile. This fact can aggravate the consequences related to their health if there is no rapid intervention once they have already fall. Nowadays researches are trying to find different solutions to solve or improve this fact since it has become a priority event that must have the quickest possible fix.

There are currently several systems intended to detect people fallings: manual alarm system, wearable sensors on devices placed in wrist or hips, video cameras recording movement, vibration based method or both, vibration and sound based methods. In this project, the device chosen consists of the combination of a digital accelerometer and a microcontroller. Fall detection detected by acceleration measurement is an area of research that can contribute and be useful in the field of human health, more specifically and as it has been commented, in security and safety of elderly people.

The main idea of the system of fall detection is to capture the accelerometer changes by means of walking and falling and then analyse them in order to set if a fall or a normal step has occurred. Data generated by the accelerometer is collected and sent via SPI protocol into a data acquisition device, the microcontroller, a card which processes and transmits all the data via Wi-Fi for analyse it in a manner that to define models able to create a falling system based on acceleration measurements.

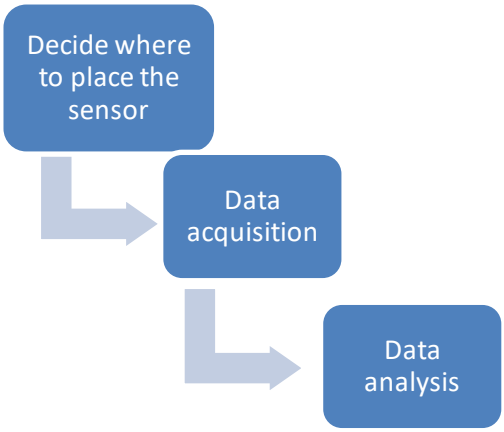
The efficiency of these tests carried out by the particular falling system can be defined by several factors:

- Sensor selected.
- Precision and accuracy of the sensor.
- Location of the sensor.
- Effectiveness of the codes used to perform the data obtained through the acquisition systems.

Therefore, the first thing to do is to decide where to place the accelerometer in order to recollect accurate data in such a way that little movements could be detected. For that, sampling frequency and the code implemented play a very important role in this acquisition.

1.1 Goals of the project

The aim of this thesis is to develop a system for detect involuntary falls, using a 3D accelerometer sensor with adequate accuracy and precision. The steps taking are:



Schema: Layout of the system

The first objective of this project is to stablish a place where to set the accelerometer. There are several possibilities available according to the kind of accelerometer chosen. Once decided the appropriated place for this purpose, this thesis has also the goal of setting two different models of counting steps, so that they can be implemented in fall detection systems.

1.2 Structure of the document

This document is structured as follows. *Chapter 2* gives an overview of the hardware used, the software implemented for acquiring and analysing data. Also talks about the expected results and the important point of placing the sensor. *Chapter 3* discusses the design of the experiments, how to implement the best physical solution. *Chapter 4 describes the two models proposed for the step detection. Chapter 5* describes all the tests carried out. *Chapter 6* discuss the results obtained in chapter 5. Finally, conclusions are presented in *Chapter 7*.

2 HARDWARE

The hardware used in this project (Figure 1) consists of the accelerometer chosen (1), the microcontroller (2), some data cables to connect the two devices (3), an external battery to power the Arduino board, a computer with software of both installed (4), Arduino and MATLAB, and finally a data cable (5) for compiling the Arduino code into the Arduino board.

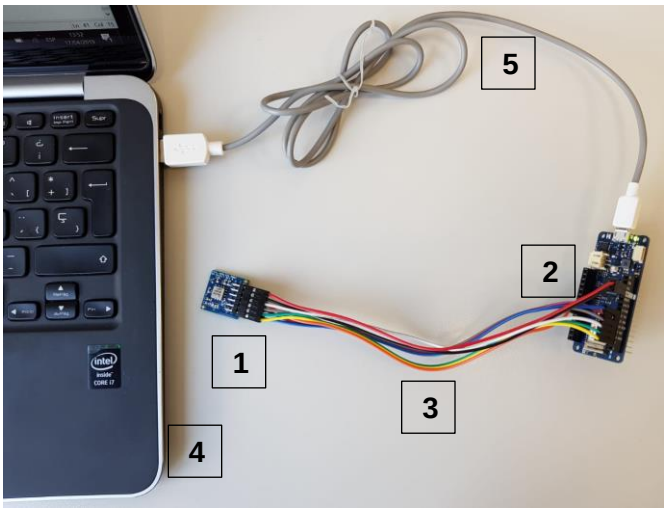


Figure 1: Hardware

2.1 Sensor

The accelerometer is the measurement instrument chosen in this project for measure the acceleration of the fall. Accelerometers are inertial devices that are used to measure changes in gravitational acceleration, across the three orthogonal axes (x, y and z). The measurements obtained are output as an analog signal, measuring in g-forces (g), where 1 g in Earth equals to 9.81 m/s^2. [1]

In order to select an appropriate sensor for the system, it should have a list of requirements that has to accomplish.

- The first one is a limitation on the sensor **size and mass**. The accelerometer must be as light and small as possible.
- The second requirement is that the accelerometer must be **ultralow noise** in order for the data to be usable.

These three requirement leave MEMS accelerometers as the only option. They provide great noise characteristics and they meet the size, mass and power requirements.

2.1.1 MEMS accelerometer

Microelectromechanical systems (MEMS) have been used to implement inertial sensing methods on an integrated chip (IC) in micro scale [2]. MEMS sensors can be used to measure physical parameters such as acceleration in a vary range of applications, such as airbag deployment, earthquake detection and navigation purposes. MEMS sensors are

smaller and lower cost than a conventional accelerometer, but they are limited in performance. Due to this facts, they have also been more and more used in smartphone, home uses and medical applications.

Principle of operation

The ADXL355 is MEMS capacitive sensor. This means that this device measures vibration or acceleration when the movement of a structure or an object occurs. The force generated by this, causes the internal mass to compress the piezoelectric material, generating an electrical charge that is proportional to the force applied on it. The fact that the load is proportional to the force and that the mass is constant, makes that the charge is also proportional to the acceleration. This makes this type of accelerometers to have higher range but low sensitivity devices than the variable capacitive accelerometers. This concept is illustrated in Figure 2:

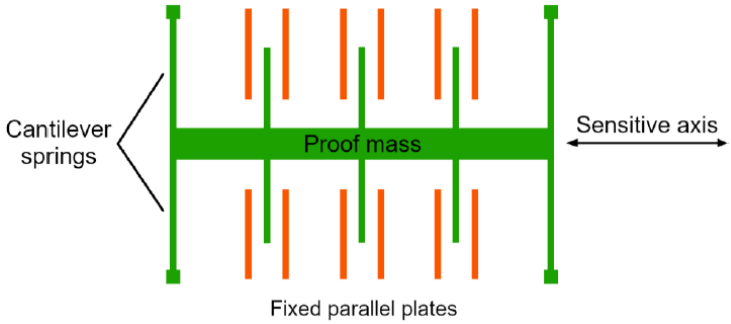


Figure 2: MEMS principle of operation

The resonant or natural frequency is defined by the combination of the stiffness of the cantilever springs and the mass of the proof mass. MEMS sensors are generally designed to have high resonant frequencies by using a lightweight proof mass and high stiffness cantilever springs. The natural frequency of the ADXL355 is at about 2.5 kHz.

2.1.2 Characteristics ADXL355

The measurements are performed by an ADXL355 accelerometer, manufactured by Analog Devices [3]. The ADXL355 is a MEMS three-axis accelerometer, ultranoise and ultrastable offset accelerometer with 3.3 V of power supply and with the possibility of select different measuring ranges. The sensor comes with an Evaluation Board as shown in the *Figure 3.*

The ADXL355 is programmable for ±2 g, ±4 g and ±8 g as an output full-scale range, in this project the chosen range is 2g. It also has a high long-term stability, allowing precision applications with minimal calibration and reducing the cost of resources and calibration, The accelerometer has a digital filter from 1 Hz to 1 kHz, with a declared consumption of 200μA. All these parameters can be found in the table 1 of the annex.

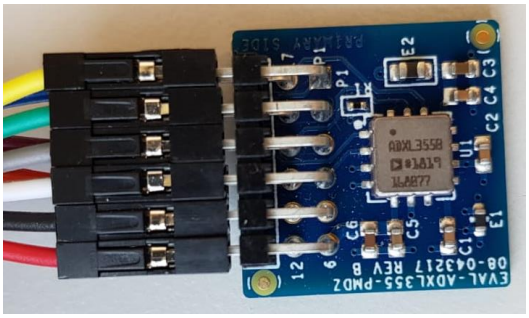


Figure 3: Accelerometer ADXL355

The accelerometer is able to measure both static acceleration from gravity and dynamic acceleration, resulting from movement and vibrations.

The sensor is equipped with 12 pins as it is shown in the *figure 4*. In the table 1 are shown the pins and their own operation. All this information is available in the table 2 of the annex. The connection with the microcontroller will be shown later.

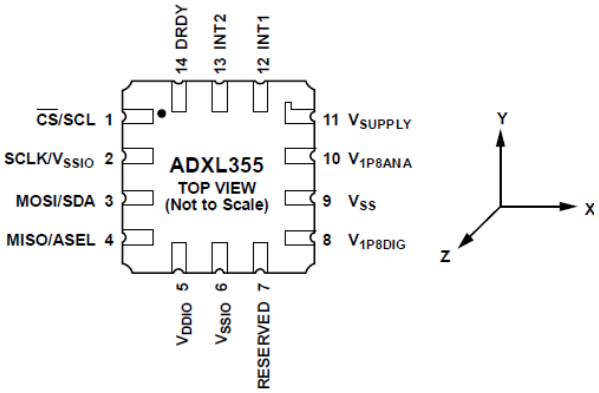


Figure 4: Accelerometer - Pin Configuration [4]

Pin	Pin function	Pin name	Pin	Pin function	Pin name
1	Chip Select	**CS**	**7**	Interrupt 1	**INT1**
2	Master Out Slave In	**MOSI**	**8**	Not Connected	**NC**
3	Master In Slave Out	**MISO**	**9**	Interrupt 2	**INT2**
4	Serial Clock	**SCLK**	**10**	Data Ready	**DRDY**
5	Digital Ground	**DGND**	**11**	Digital Ground	**DGND**
6	Digital Power	**VDD**	**12**	Digital Power	**VDD**

Table 1: Accelerometer - Pin configuration

Normal operation

The axis of acceleration sensitivity in this accelerometer are displayed as it sets the *figure 5*. Considering the case that the accelerometer is laid down in a flat surface, as for example in the same position set in the picture and with no movement recorded, the Z-axis is in the ±1-g field of gravity and reads ±1g for being the one perpendicular to the surface, the X- and Y-axis are in the 0-g field of gravity and thus read 0g. [4]

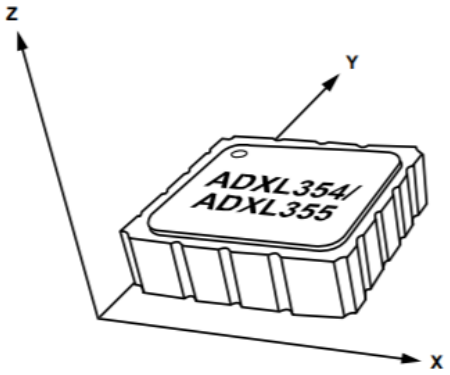


Figure 5: Axes of Acceleration [4]

Serial SPI communication

The communication between the microcontroller and the peripheral devices is handled via **S**erial **P**eripheral **I**nterface (**SPI**), which makes use of the *master/slave principle*. The **Master** is the device that controls the system. It has the ability to invade and receive data and commands to start the transmission session. It also provides the synchronisation clock of the data exchange. The **Slave** is a peripheral device that can receive and send data, but cannot send commands. In this particular case the microcontroller Arduino acts as the master device while the ADXL355 is connected as slave device. The communication is handled as shown in *Figure 6*:

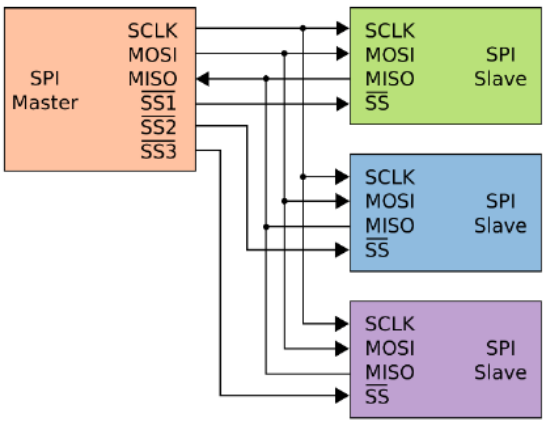


Figure 6: SPI master slave communication protocol

Typically there are three lines common to all the devices:

- **MISO** (Master In Slave Out) - The Slave line for sending data to the master.
- **MOSI** (Master Out Slave In) - The Master line for sending data to the peripherals.
- **SCK** (Serial Clock) - The clock pulses which synchronize data transmission generated by the master.

Additionally, there is one Slave Select (**SS**) line for each device, which is used to enable or disable the communication with the specific peripheral. This way, the master can choose which slave device to send data to and receive data from, even though the communication lines are shared between all of them. In order to receive data from a slave device, the master device has to send a read command to that specific device first. This prevents the different peripherals from sending data at the same time, which would lead to parts of the data being corrupted. [5]

When a device's Slave Select pin is low, it communicates with the master. When it's high, it ignores the master. This allows to have multiple SPI devices sharing the same MISO, MOSI, and CLK lines.

The schema defines the connection of the ADXL355 for SPI communication with a generic processor is the one shown in figure 7:

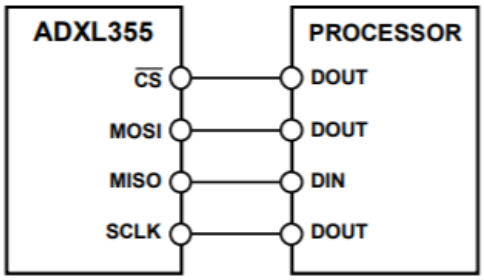


Figure 7: 4-wire SPI connection [4]

The data in XDATA, YDATA, and ZDATA (data from each axis of the accelerometer) is always the most recent available because the accelerometer continuously updates its data registers. It is not guaranteed that data form a set corresponding to one sample point in time. The routine used to retrieve the data from the device, controls this data set continuity. For multibyte read or write transactions through either serial interface, it has been used the FIFO address, so that, data can be read continuously from the FIFO as a multibyte transaction.

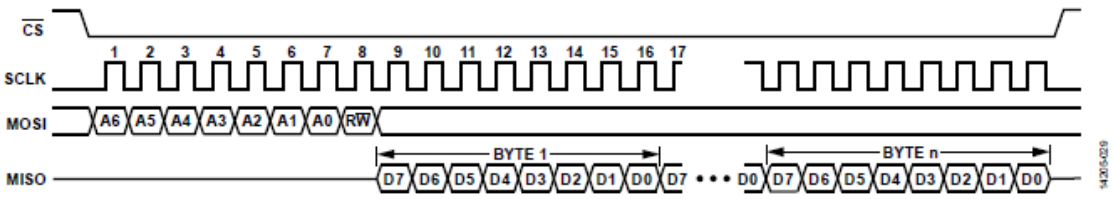


Figure 8: SPI Protocol – Multibyte Read [4]

Register configuration

This accelerometer has an output resolution of up to 24 bits for each axis: the significant information is however contained in the first 20bit (a bit with the sign), while the 4 bits less significant are discarded because affected by possible noise. The accelerations along the 3 axes (x, y and z) are stored inside 9 registers: 3 registers for each axis. Each register consists of 8 bits with a total of 24 bits for each axis, as it shown in *figure 9*.

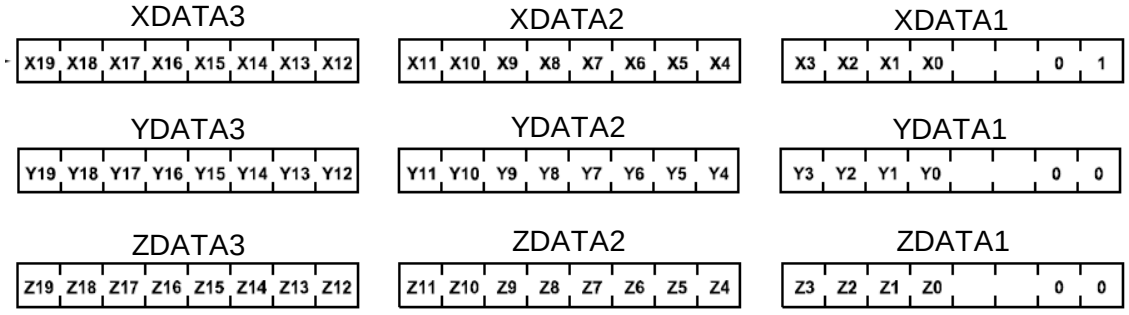


Figure 9: Register configuration

According to reading acceleration data from the interface, this is found as left justified where the register address is ordered from most significant data to least significant data, which allows to use multibyte transfers and to take only the 20 bits data required. For example, for x-axis acceleration, XDATA1 is the low byte register and XDATA3 is the high byte register.

Acceleration data is received in the so-called two's complement notation which is used to represent signed integers as binary numbers. The conversion is done by the microcontroller.

FIFO

To read acceleration data from the sensor via SPI it can be read repeatedly from the registers corresponding to the x-, y- and z-axis, however, since the acceleration data is 20 bits long while each register only contains 8 bits (1 byte) three registers have to be read to access the measurement for each axis. This poses a problem when new acceleration data arrives while the previous data is read out. In this case, it is possible that the three bytes comprising a data point originate from different measurements at different points in time. This mixing of data is undesirable since it can invalidate the measurement results. Furthermore, if the data registers are not read before the next measurement is available, the previous values will be overwritten.

However, the ADXL355 features a so-called FIFO (**F**irst **I**n **F**irst **O**ut), a data structure that works like a queue and allows data to be stored for a short amount of time before it is read. By default, the FIFO is not available. If the FIFO is in use, it operates in a stream mode, this means that when the FIFO overruns the new data overwrites the oldest one. A read from the FIFO address guarantees that the three bytes associated with the acceleration

measurement on an axis all pertain to the same measurement. The FIFO never overruns, and data is always taken out in sets as a multiple of three data points.

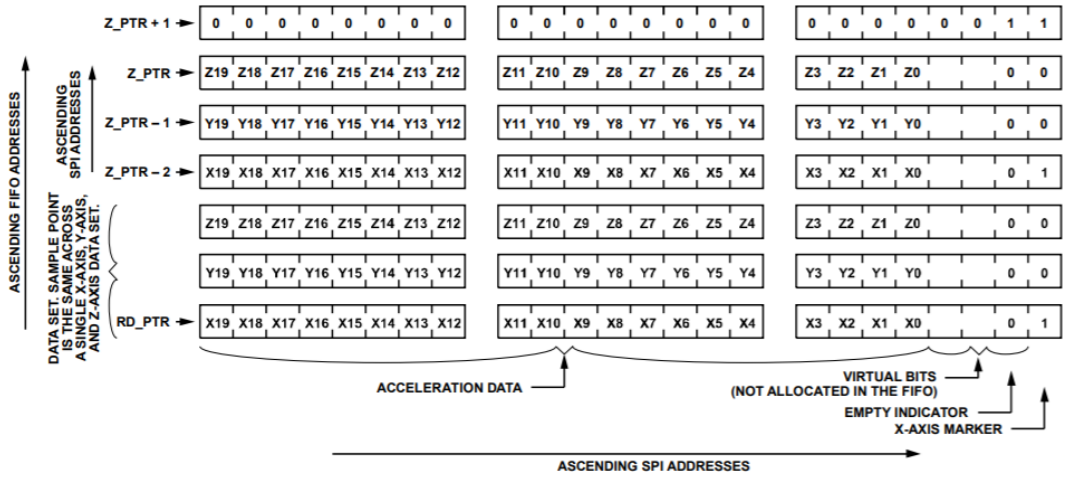


Figure 10: FIFO Data organization [4]

The *figure 10* shows that there are 96 21-bit locations in the FIFO. Each location contains 20 bits of data and a **marker bit** for the **x-axis** data, which allows a quick identification of the next samples of data. The FIFO control logic inserts the two LSB reads on the interface. Bit 1 indicates that an attempt was made to read an empty FIFO, and that the data is not valid acceleration data, as happens in the figure 10 in the first row. Moreover, bit 0 is a marker bit to identify the x-axis, which allows a user to verify that the FIFO data was correctly read.

2.2 Microcontroller

The microcontroller board that has been chosen for this project in order to arrange the communication between the accelerometer and the computer is an Arduino MKR WIFI 1010 (*Figure 11*). It is an open-platform circuit board that features a microcontroller, several programmable digital input/output pins as well as analog connectors and a low power Wi-Fi module integrated on it. [6]

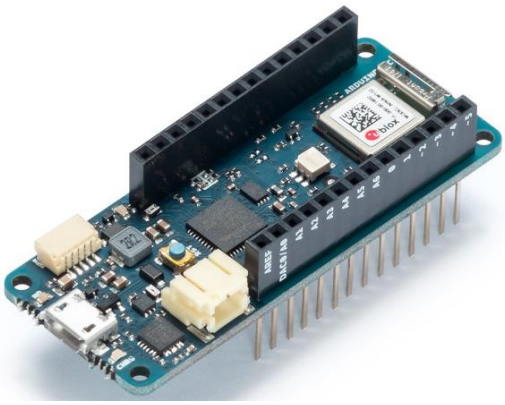


Figure 11: Arduino MKR WIFI 1010 board

The MKR Wi-Fi 1010 offers low power consumption and has been designed not only to speed up and simplify the prototyping of Wi-Fi-based IoT applications, but also to be embedded in IoT applications that require Wi-Fi connectivity, which gives the possibility to run different types of projects.

Its USB port can be used to supply 5 V of power to the board. The MKR1010 board can be programmed via USB port through the Arduino development environment (IDE). The programming language used in this is based on the C/C++ language with a set of particular functions of Arduino environment.

All of these features plus the small size and low energy consumption make this board the preferred choice for this thesis, which is a perfect example of the emerging IoT battery-powered projects.

2.3 Connectorization

The pin connection between accelerometer and microcontroller is set in the *table 2*. This configuration allows setting the SPI communication between them.

ADXL355 Pin Number	Pin description	Arduino Pin Number
1	Chip Select (CS)	07
2	Master Out Slave In	08
3	Master In Slave Out	10
4	Serial Clock (CLK)	09
5 / 11	Digital Ground	GND
6 / 12	Digital Power	VCC
7	Pin interrupt	INT1
9	Pin interrupt	INT2
10	Data Ready	DRDY

Table 2: pin connection between microcontroller and Accelerometer

In this configuration the role of *master* is performed by Arduino while that of *slave* by the accelerometer. The master, in this case, accesses to the registers of the slave by sending a byte, where the first 7 bits represent the address of the log to access while the last bit represents the access mode (Read or Write). A scheme of the connection is shown below:

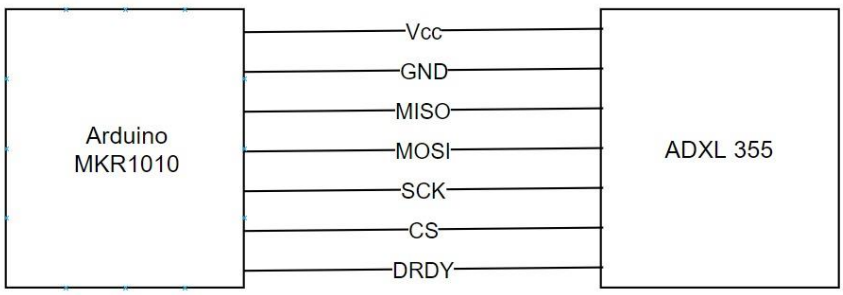


Figure 12: Scheme of connection between microcontroller and accelerometer

The final and real wiring connection between both devices is shown in the *figure 13*.

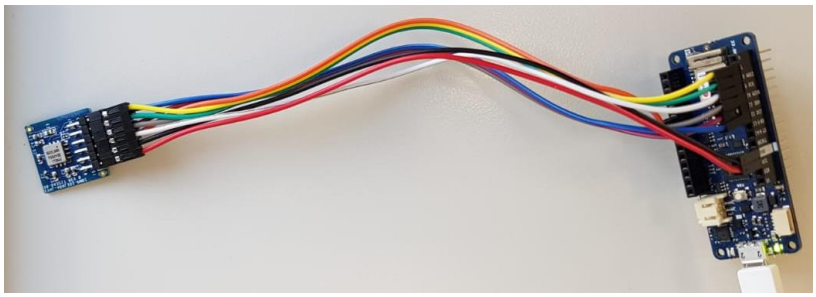


Figure 13: pin connection between accelerometer and microcontroller

The interconnection between the accelerometer and the Arduino board has been following the pin configuration given in the datasheet of the ADXL355 accelerometer. This can be found in the table 2 of the annex .

Power supply

Talking about how to power the system, there are several ways which can be used:

- By the grid throughout a power converter.
- By a computer via USB port
- By an external battery.

Finally, for the kind of application on development, it has chosen a 5V wearable power bank to power the microcontroller, in order to make it much easier. It can thus provide an output of 3.3V to power the accelerometer.

This picture shows an idea of how the system is interconnected.

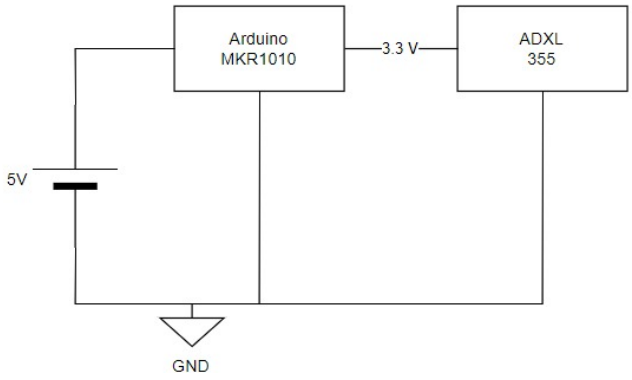


Figure 14: power supply scheme

It also has a Li-Po battery that allows the Arduino board to run on both battery power and an external 5 V source, running on external power while charging the Li-Po battery, switching from one to another automatically.

3 SOFTWARE

3.1 Acquisition data

The open-source Arduino Software (IDE - Integrated Development Environment) is a cross-platform application perfect for many operating system. It is used to write and upload programs to Arduino compatible boards, in this particular case, the MKR WIFI 1010 board [7].

The code implemented for this project includes some relevant functions so that it is tried to collect data sent via SPI from the sensor in the most efficient way. These are described below:

Parameters defined

In order to be able to operate with Arduino software, it is necessary to include, that is, to define in the code the libraries which it is required to work with. The accelerometer data is passed to Arduino via the SPI protocol, using the **SPI** library while the **WiFiNINA** library is used for Wi-Fi communication protocol. It is also necessary an **ADXL355** library used for defining specific parameters of the accelerometer.

SPI function

Before acquiring any data it is required to set the SPI communication protocol between the two devices. In order to receive data from the slave device, the master device has to send first a read command, so that the communication will be stablished.

```
/* Read registry in specific device address */
unsigned int readRegistry(byte thisRegister) {
  unsigned int result = 0;
  byte dataToSend = (thisRegister << 1) | READ_BYTE;

  digitalWrite(CHIP_SELECT_PIN, LOW);
  SPI.transfer(dataToSend);
  result = SPI.transfer(0x00);
  digitalWrite(CHIP_SELECT_PIN, HIGH);
  return result;
}

/* Read multiple registries */
void readMultipleData(int *addresses, int dataSize, uint32_t *readedData) {
  digitalWrite(CHIP_SELECT_PIN, LOW);
  byte dataToSend = (0x08 << 1) | READ_BYTE;
  SPI.transfer(dataToSend);
  for (int i = 0; i < dataSize; i++) {
    readedData[i] = (uint32_t) SPI.transfer(0x00);
  }
  digitalWrite(CHIP_SELECT_PIN, HIGH);
}
```

Illustration 1: Code Arduino - SPI protocol settings

While the SPI communication is defined, the main read function *readMultipleData* is also defined, essential to acquire data from the accelerometer.

ReadMultipleData

The fundamental part of the firmware is contained within the function *adxl355ISR()*, an interrupt function which will be commented further later. Inside this function there is another function called ***readMultipleData*** function, which is used to read data from the 9 registers that the sensor provides. This function takes as arguments the addresses of the registers (axisAddresses), the number of registers (dataSize) and the measured value of the axis (axisMeasures), defined as it shows the *illustration 2*.

```
/*READ ACCELEROMETER*/
int axisAddresses[] = {XDATA3, XDATA2, XDATA1, YDATA3, YDATA2, YDATA1, ZDATA3, ZDATA2, ZDATA1};
uint32_t axisMeasures[] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
int dataSize = 9;

// Read accelerometer data
readMultipleData(axisAddresses, dataSize, axisMeasures);
```

Illustration 2: Code Arduino – readMultipleData

Since the SPI.transfer() function transfers 8 bits of *uint8_t* type at a time, the values read by the SPI protocol are converted directly to uint32_t (no sign). The three registers (one for each axis) are grouped in a single register (e.g. axdata for the axis x). At this point the data is converted into an integer with a 32 bit plus sign, this is why a right shift of 12 bits is required, deleting the 4 less significant bits.

```
// Split data
uint32_t Xdata = (axisMeasures[0] << 24) + (axisMeasures[1] << 16) + (axisMeasures[2] << 8);
uint32_t Ydata = (axisMeasures[3] << 24) + (axisMeasures[4] << 16) + (axisMeasures[5] << 8);
uint32_t Zdata = (axisMeasures[6] << 24) + (axisMeasures[7] << 16) + (axisMeasures[8] << 8);

int32_t axdata = (int32_t) Xdata;
int32_t aydata = (int32_t) Ydata;
int32_t azdata = (int32_t) Zdata;

axdata = axdata >> 12;
aydata = aydata >> 12;
azdata = azdata >> 12;
```

Illustration 3: Code Arduino - shift data

WifiServer

Once set a specific port number to make the Wi-Fi connection through it, it is possible to stablish the WifiServer connection between Arduino (Server) and MATLAB (Client). The connection takes part inside the main loop.

For making this connection possible, it is required that Arduino not only sends the data to MATLAB but also reads what the Client sends to the Server, in order to stablish a secure connection and not to lose it. This is a Quality of Service level 1.

```
void loop() {
  WiFiClient client = server.available();
  digitalWrite(LED6,HIGH);
  if (client) {
    boolean currentLineIsBlank = true;

    attachInterrupt(digitalPinToInterrupt(ADXLINT), adxl355ISR, FALLING);

    while (client.connected()) {
      char c = client.read();
      while (FIFO_WrittenBytes(&fifo_buffer) < 1040) {
        ;
      }
      /*Envio de los 1040 bytes*/
      byte transmit_Buffer[1040];
      for (int i = 0; i < 1040; i++) {
        transmit_Buffer [i] = (byte) FIFO_Get(&fifo_buffer);
      }
      contator = contator +1;
      digitalWrite(TRANSMIT_LED,HIGH);
      client.write (transmit_Buffer, 1040);
      digitalWrite(TRANSMIT_LED,LOW);
      //delay(1);
    }

    // close the connection:
    client.stop();
    contator=1;
    detachInterrupt(digitalPinToInterrupt(ADXLINT));
    FIFO Init (&fifo buffer, data store, buff size);
```

Illustration 4: Code Arduino - main loop

Whenever the connection between server and client is stablished, a LED in the Arduino board switches on. This is because all the test are been carried out with an external battery, so that, the test can only start when this LED is on; also for making sure that the connection does not get lost during all the tests.

ISR function

As it has been mentioned previously, it has been used an interrupt function. This is because using a delay function normally suppose a continuous consumption of the processor and energy, having to ask continuously for the status of the input. However, the **interruption** allows to associate a function to the occurrence of a certain event. This associated call-back function is called ISR (Interruption Service Routine). [8]

The one used in this project is called *adxl355ISR()*. The ISR function is used for the *attachInterrupt*. Interrupts are useful for making readings happen automatically in the microcontroller until having 1040 bytes, in order to send packets of 1040 bytes each time, once the connection is stablished. Each packet is composed of 65 acquisitions of: 4 bytes of the counter, 4 bytes of X data, 4 bytes of Y data and 4 bytes of Z data. The counter is simply used to make sure that all the information is sent and it has been sent in the correct order.

For making this attach possible it is required to define the appropriate pin, in this case pin number 0.

To implement the ISR function it is needed a First In First Out function (FIFO) in order to send packets of 1040 bytes to MATLAB. So, once the ISR connection is stablished, whenever one packet is sent, the following packet is created, in such a way that, when the first packet is already sent the second one is already ready for being sent too. With this method, waiting times are deleted.

```
//Reading XYZ
uint8_t tmp0 = (uint8_t) axdata & 0x00000000000000FF;
uint8_t tmp1 = (uint8_t) (axdata >> 8) & 0x00000000000000FF;
uint8_t tmp2 = (uint8_t) (axdata >> 16) & 0x00000000000000FF;
uint8_t tmp3 = (uint8_t) (axdata >> 24) & 0x00000000000000FF;
uint8_t tmp4 = (uint8_t) aydata  & 0x00000000000000FF;
uint8_t tmp5 = (uint8_t) (aydata >> 8) & 0x00000000000000FF;
uint8_t tmp6 = (uint8_t) (aydata >> 16) & 0x00000000000000FF;
uint8_t tmp7 = (uint8_t) (aydata >> 32) & 0x00000000000000FF;
uint8_t tmp8 = (uint8_t) azdata  & 0x00000000000000FF;
uint8_t tmp9 = (uint8_t) (azdata >> 8) & 0x00000000000000FF;
uint8_t tmp10 = (uint8_t) (azdata >> 16) & 0x00000000000000FF;
uint8_t tmp11 = (uint8_t) (azdata >> 32) & 0x00000000000000FF;
//Reading contator
uint8_t c1 = (uint8_t) contator  & 0x00000000000000FF;
uint8_t c2 = (uint8_t) (contator >> 8) & 0x00000000000000FF;
uint8_t c3 = (uint8_t) (contator >> 16) & 0x00000000000000FF;
uint8_t c4 = (uint8_t) (contator >> 32) & 0x00000000000000FF;
noInterrupts();
```

```
FIFO_Put(&fifo_buffer, c1);
FIFO_Put(&fifo_buffer, c2);
FIFO_Put(&fifo_buffer, c3);
FIFO_Put(&fifo_buffer, c4);
FIFO_Put(&fifo_buffer, tmp0);
FIFO_Put(&fifo_buffer, tmp1);
FIFO_Put(&fifo_buffer, tmp2);
FIFO_Put(&fifo_buffer, tmp3);
FIFO_Put(&fifo_buffer, tmp4);
FIFO_Put(&fifo_buffer, tmp5);
FIFO_Put(&fifo_buffer, tmp6);
FIFO_Put(&fifo_buffer, tmp7);
FIFO_Put(&fifo_buffer, tmp8);
FIFO_Put(&fifo_buffer, tmp9);
FIFO_Put(&fifo_buffer, tmp10);
FIFO_Put(&fifo_buffer, tmp11);
interrupts();
```

Illustration 5: Code Arduino - ISR FIFO function

The code also includes a counter so that an account of the data acquisition takes place in order to know if some data has been lost during the sending. Every packet (1040 bytes) is made of 65 data acquisition: 32 bits each axe (4 bytes each, 12 bytes in total) plus 4 bytes for the counter.

3.2 Data analysis

Data analysis consists in two different models of filtering the input signal in order to obtain two possible solutions for the system. For analysing data, it has used two scripts in MATLAB. The main script and a function. Father later the two different models will be explained.

The connection client server in MATLAB is done by a function called **tcpip**, where it is mandatory to set the port server (the same chosen in the Arduino code) and the Remote host ID, the same that the Arduino board has been connected to.

```
%% TCP CLIENT
tcph = tcpip('192.168.1.6',80,'NetworkRole','client',...
                'BytesAvailableFcnMode', 'byte',...
                'BytesAvailableFcnCount',nbytes,...
                'BytesAvailableFcn','readtcp',...
                'InputBufferSize', 100000,...
                'ReadAsyncMode', 'continuous');
```

Illustration 6: Code MATLAB - tcp client

It is important to open but also to close this function. The closure has to be once the HDF5 file has been written. Another crucial thing is to send something to the server (write), in order to stablish the connection. In this case it is just send a row of numbers that the Arduino receives and enables the connection.

```
fopen(tcph);
fwrite(tcph,uint8([0,1,2,3,4]));
```

Illustration 7: Code MATLAB - write for connectivity

While stablishing this connection, there is need to create an HDF5 file in the main function. This kind of file formats are made to designed to store and organize large amounts of data, perfect for the current application.

```
%% HDF5 CREATE
cont_data = 1;
path = ['Data/Walking_Data' num2str(cont_data) '.h5'];
A = exist(path,'file');

while A~=0
    cont_data = cont_data + 1;
    path = ['Data/Walking_Data' num2str(cont_data) '.h5'];
    A = exist(path,'file');
end

h5create(path,'/DS_data',[Inf num_colum],'ChunkSize',[nbytes num_colum],...
            'Datatype','double');
```

Illustration 8: Code MATLAB - create HDF5 file

The creation of the files consists in first check the presence of files in the folder they are kept and then create the *.h file* with the next number unused. This function is created in order to create the next *Walking_dataX.h5 file* automatically.

The program is set to finish with a specific time frame. If Arduino is set to read with a frequency of 250 Hz, having 1040 bytes each packet by 16 bytes each acquisition, the time required for a single packet is 2.6 seconds. However, in every cycle it is received a number of 15 packets. That is why the tests are set in multiple of 3.9 seconds. In all the tests carried out it has been almost 2 minutes and a half.

```
while (ishandle(H) && toc<(3.9*35) )
    pause(0.0005);
end
```

Illustration 9: Code MATLAB - timing tests

All mentioned since now belongs to the main script. However, it has been used a function script in order to implement a function which reorder de input data and stores it in a specific format. In this case the function is called *readtcp().*

```
disp('ACQUIRING\n');
data_aquired = fread(tcph,nbytes,'uint8');

% Convert into a matrix
data_aquired_mat = uint8(vec2mat(data_aquired', 4));
% Convert into columns
data_aquired_row = arrayfun(@(n) converti(data_aquired_mat(n,:)), (1:nbytes/4));
% Reshape
data_aquired_reshape = reshape(data_aquired_row,4,[]);
% Transpose
data_aquired_colum = data_aquired_reshape';
```

Illustration 10: Code MATLAB - input data format

The input data is read in a single column. For being able to operate with data is mandatory to reorder it in order to, at the end, have a matrix based on 4 columns: the counter, x data, y data and z data respectively. This is what shows the *image 10*, where several steps have been taken to get this matrix:

- Data is set in such a way that every 4 bytes is one value. The first step consist in making a matrix made of 4 columns, representing each row a value.
- Second step consists in cluster this four columns in one, obtaining the final value. So it is obtained again a matrix formed by only one column.
- It is need now a reshape of the data from a single column to a matrix of 4 rows: counter, x y and z.
- Finally, transpose this matrix to obtain the final one made of 4 columns.

Once the matrix is obtained, it has been chosen a FIFO to set a secondary buffer. In the pc there are four processors, if all of them are being used it is impossible to read and write at the same time. That is why a secondary buffer is used.

This is shown in the *illustration 11,* where B1 and B2 represent the two different functions, while one is reading the other one is writing in the file.

```
    if (~condition_buffer)
        B1((1+c*(nbytes/16)):((c+1)*nbytes/16),:)=data_aquired_colum;
        c = c + 1;
    else
        B2((1+c*(nbytes/16)):((c+1)*nbytes/16),:)=data_aquired_colum;
        c = c + 1;
    end

    if (c > 14)
        c=0;
        condition_buffer = (~condition_buffer);
        if (condition_buffer)
            h5write(path,'/DS_data',B1,[num_rows*h+1 1],[num_rows 4]);
        else
            h5write(path,'/DS_data',B2,[num_rows*h+1 1],[num_rows 4]);
        end
    h = h + 1;
    end
end
```

Illustration 11: Code MATLAB – secondary buffer

4 DESIGN OF THE EXPERIMENTATION

4.1 Expected results

The efficiency of the tests that should be carried out could be defined and conditioned by several factors, among others:

- The sensor selected.
- The place where the sensor is going to be placed.
- Effectiveness of the code used to perform the data obtained through the acquisition systems.
- Effectiveness of the analysis code.
- The way user walk.

Talking about the expected results, it is known that if we placed the sensor horizontally on a flat surface with no vibration, we would get a result like the one shown in figure X, in which Z axe is 1g and the other two axis, X and Y are 0g. Assuming that the sum of the three axis is always 1g.

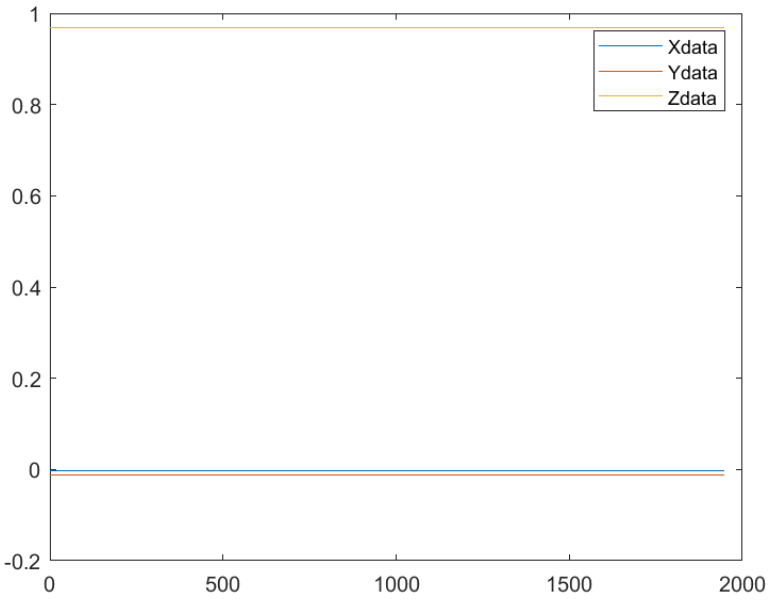


Figure 15: no motion of the sensor

The same would happen if it is changed the sensor layout, this means if the orientation is changed. In that case, it would be any of the other two axes, X or Y, which would be 1g.

However, whenever steps are carried out, the graph changes dramatically (figure 16). The three axes will capture movement so that, as already mentioned, the sum of the 3 is 1g. The expected layout of the three axis is the one shown on the figure.

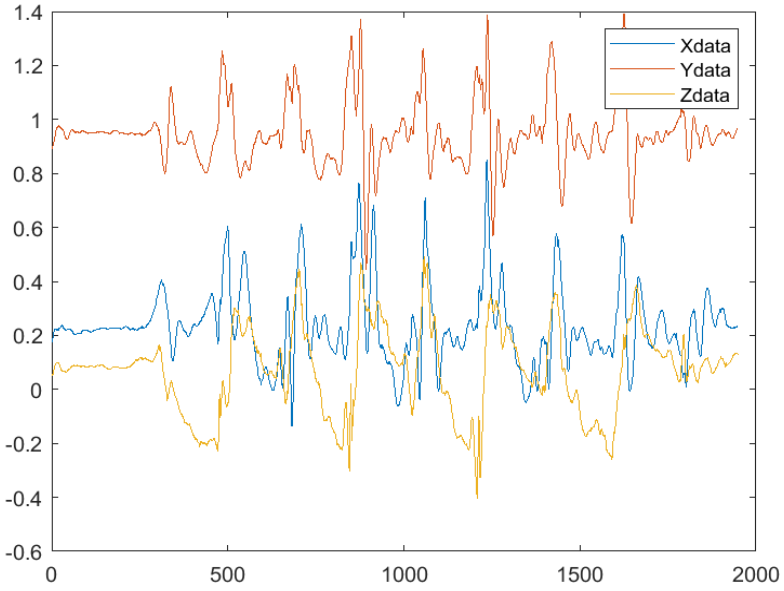


Figure 16: step recording of the sensor

4.2 Position of the sensor

The most important part of this thesis is to establish where the sensor is going to be located, since the filtering of the input signal changes depending on this. To know where to place it, once the data acquisition is done correctly and the system is able to analyse the first samples, the first tests are performed.

The first test are carried out by placing the sensor in the **ground** (figure 17). The accelerometer is fixed in the floor, so that when the event occurs, the vibrations generated in the ground next to it are captured and are supposed to be read and analysed by the accelerometer.

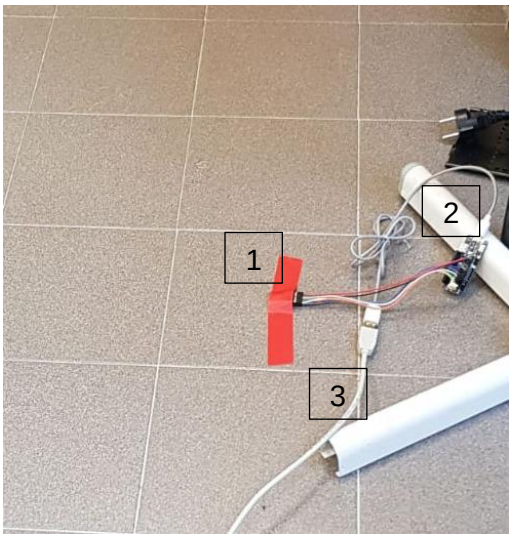


Figure 17: Placement 1: floor

In the image above, the sensor is fixed in the floor with a tape (1), trying to place the sensor as flat as possible in the ground. Connected to it is the microcontroller (2) and the system is feed with an USB port from the computer (3).

Once the stage is set, the first steps are performed, testing at various distances of the sensor and with different intensities (it is different strength in the steps), so that all possible possibilities are covered. We find a situation (shown in figure 18) in which the information collected is very weak in both cases. These cases are two different assumptions, each with a different sampling frequency to try to solve the problem we encounter. It turns out that the accelerometer is unable to pick up the vibrations because the ground on which it is located is much thicker than expected, so the wave propagation is very weak.

As it has been mentioned, it has been tried to improve the sampling frequency to try to make the sensor as sensitive as possible, but it has not been fixed. Figure 18 shows that only the steps vary (in z axis, perpendicular to the ground) from 0 to 0.001. Values that are too poor to be able to operate with.

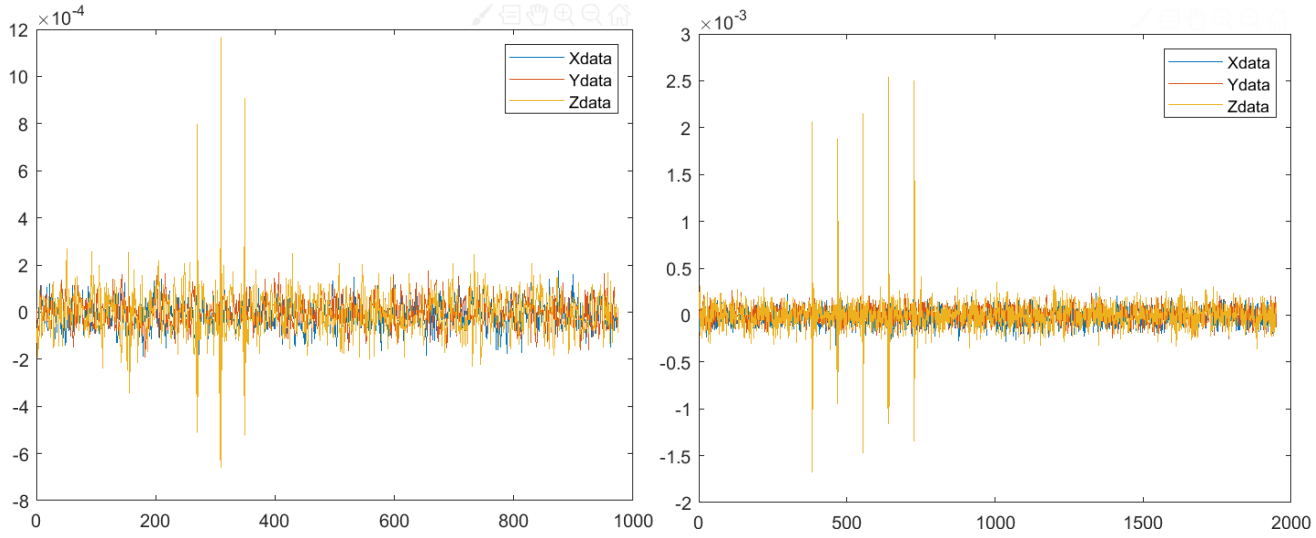


Figure 18: First tests with sensor placed on the ground

The reason why this fact happens is because the sensor is not precise enough to be able to catch the vibrations of a ground which is apparently much thicker than it was expected to be, so the vibrations propagated through the ground around it are not good indicators enough to measure steps.

Only in the vicinity of the sensor, very close to it, it is possible to get a correct reading of the steps, however, whenever the steps are carried out at a distance greater than half a metre, it is impossible to capture anything. This is why placing the sensor on the ground has been discarded.

The second option for this project is setting the same accelerometer in a **belt** placed in the waist, since it is thought that by putting the sensor on the waist, with the movement of walking, the information can be read in a more precise way. In the same way as in the previous case, the first experiments have been carried out in order to determine whether this second arrangement is suitable for this system.

By setting an initial sampling rate of 250 Hz, it can be seen from figure 19 that the results obtained are much more realistic, in comparison with the comments made in the section on expected results. In this second graph, the three distinct axes appear on a scale appropriate to the reality.

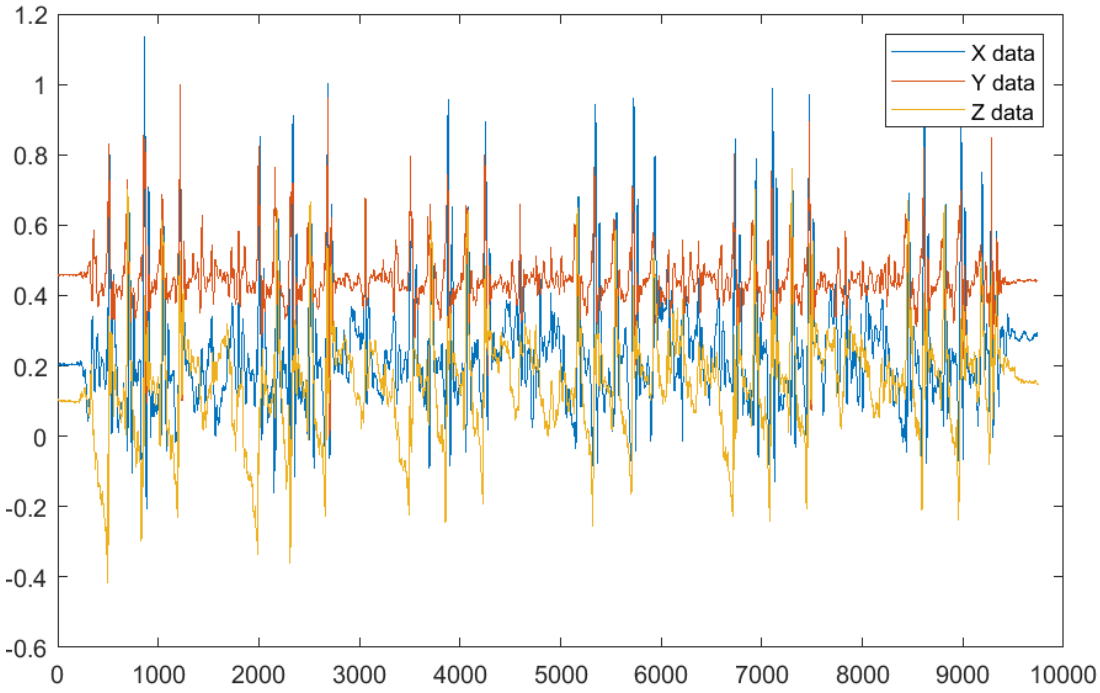


Figure 19: first steps with the sensor placed in the waist

So, as it has seen, by positioning the sensor on the waist, is the best option of the two proposals. This is why, from this moment on, the algorithm models that are going to be implemented for the study of the steps, will be in operation of this provision.

Finally, an important aspect to consider is which of the three axis is the one to study. As shown in figure 20, the arrangement of the accelerometer in the belt indicates that the **Y axis** is the one that follows with the advance of the steps. Therefore it is the Y axis the one which will be studied.

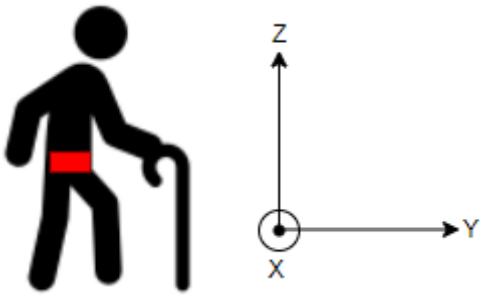


Figure 20: disposition of axes in the belt

4.3 System

The diagram of the final system is illustrated in the figure 21. Formed by a user who carries on his waist the belt with the *portable data acquisition system* and with the connection Wi-Fi established between this system and the computer so that the information can be sent whenever required.

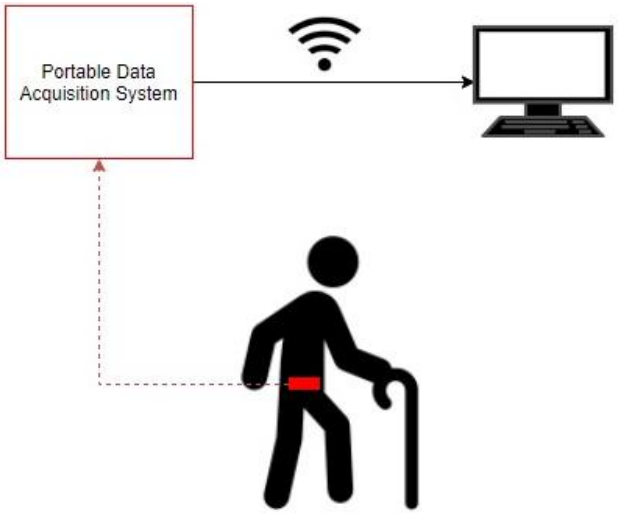


Figure 21: Diagram of the overall accelerometer system layout

The system consists of five sections connected, this is shown in the figure 22:

1. The accelerometer sensor, placed just in the down part of the belly bottom.
2. The signal processing and data acquisition circuit, the microcontroller.
3. The external battery for powering the system.
4. The belt, made with a strap and incorporates a manual closure.
5. The wires connecting the accelerometer sensor to the signal processing circuit.

Figure 22: real model

4.4 Scenario

The scenario in which the tests have been carried out is what shows in figure 23. It consists of a straight line formed by six stickers, all separated by the same distance of approximately one human step. Each step is marked by a square of red tape, so that every step taken must be on top of it.

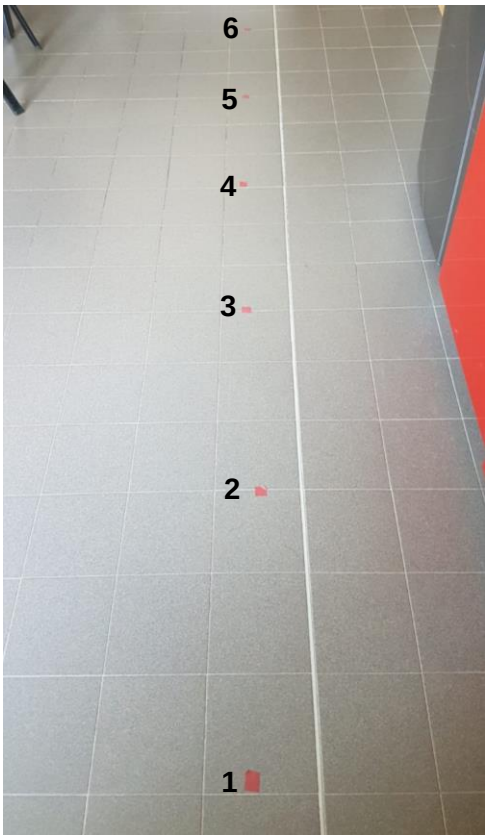


Figure 23: layout

This scenario is delimited by the form and installation of the laboratory where they were made. Although it may be thought to be an impediment, it is considered an advantage because in this way the rotation of the sensor can be also studied.

4.5 Description of the tests

The experiments have been carried out in the scenario shown above. The tests consists in walking on the red tape a total of 5 steps until the end is reached. Once reach the last piece of tape, there is need to make a 180º turn to return for the same direction but in the opposite direction. Tests have been carried out by two testers of different sex, different body weight and different kind of walking. Simply the tester has to wear the belt, run the program and place in the first square of red tape. The code includes a “bong” sound both at the beginning and at the end of the acquisition. Whenever it sounds at the beginning, the person has 5 seconds to place and start walking when it sounds. Same occurs at the end, when it sounds the person has to stop walking.

5 MODELS OF DATA ANALISYS

To carry out the models of the algorithms designed for this thesis, a first test recording of about 6 times the scenario (made of 5 steps). So the filtering of the input signal for both models is based on this file formed by a total of 30 steps. This file is *Walking_Data1.h5* (illustration 12).

Once chosen the file to study, simply there is need to open it and read the matrix formed by the four columns: the counter, X data, Y data and Z data, as it has been explained in the chapter *2.3 Data analysis.*

```
%% H5 OPEN
file = H5F.open('Walking_Data1.h5');
%% H5 READ
data=h5read('Walking_Data1.h5','/DS_data');
```

Illustration 12: Code MATLAB - open and read h5 file

The first step is to perform the conversion from bits to g, for that, having chosen a full of scale of 2g, so that the sensor is much more precise for this application, the result of the conversion in shown in illustration 13. It is done for the three axis, however, it is only mandatory on the Y axe, as it is the important one. *Y* has a size of 9750x1.

```
%OBTEINING DATA
X = data(:,2).*2./(2^19-1);
Y = data(:,3).*2./(2^19-1);
Z = data(:,4).*2./(2^19-1);
```

Illustration 13: Code MATLAB - obtaining data from axes

These first two steps are common to both algorithms, from now on, both are defined separately so that each of them will have a different filtering of the input signal.

The difference between them is that in the first one we look for a square signal that encompasses each block of 5 steps, so that after obtaining these blocks, they can be studied separately. However, in the second algorithm, the square signal is made from each of the peaks representing the steps. This will become clearer in the following pages.

5.1 Algorithm model 1

In this first model, two different filtering have been carried out. They will be described below. First of all, the input signal has been normalized:

```
Y=normalize(Y,'range');
```

Illustration 14: Code MATLAB algorithm 1 - first normalization

In order to have the signal between 0 and 1, as shown in the following image (red graph), it has used the MATLAB function *normalize* with the input argument *range*, which allows to set this range of values. The original input signal of the Y axe is the blue one.

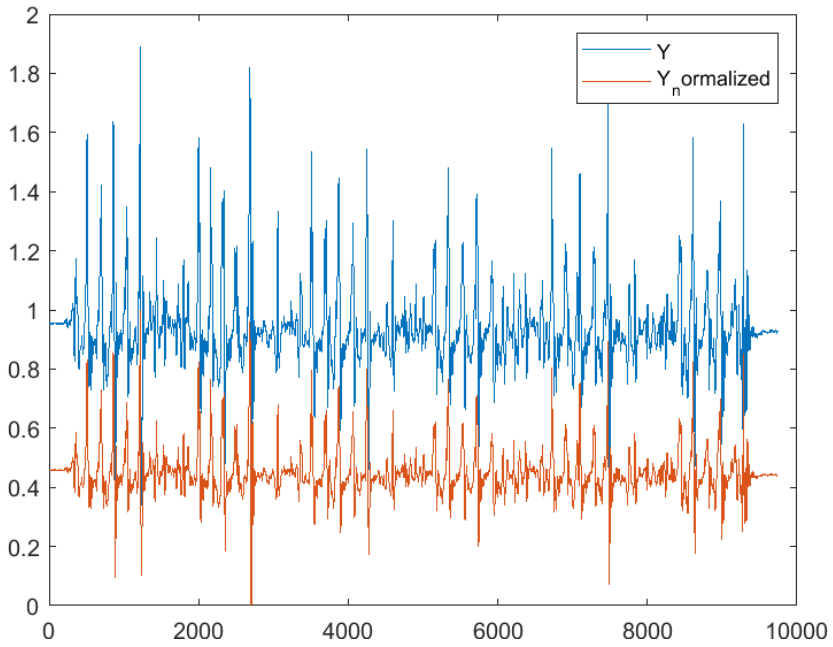


Figure 24: Algorithm 1 – Blue: input signal / Red: Normalized signal

The next step is to apply the first filtering: a bandpass filter, which allows the signal between two specific frequencies to pass, but that discriminates against signals at other frequencies, without distorting the input signal or introducing extra noise. This band of frequencies can be any width and is commonly known as the filters bandwidth, composed by the lower cut-off frequency and the higher cut-off frequency that in this particular case is 1.1 Hz and 2 Hz. [9].

```
%FIR bandpass filter, entre 1.1 y 2 Hz
bpFilt = designfilt('bandpassfir',...
         'FilterOrder',2000, ...
         'CutoffFrequency1',1.1,...
         'CutoffFrequency2',2, ...
         'SampleRate',250);
bp_Y = filtfilt(bpFilt,Y);
```

Illustration 15: Code MATLAB algorithm 1 - Bandpass filter

Bandwidth is commonly defined as the frequency range that exists between two specified frequency cut-off points (fc), that are 3dB below the maximum centre or resonant peak while attenuating or weakening the others outside of these two points. This can be seen in the following image.

This first filter is carried out in order to isolate the 30 steps carried out, so that it can operate much better with the signal of illustration 3 in comparison to illustration 1.

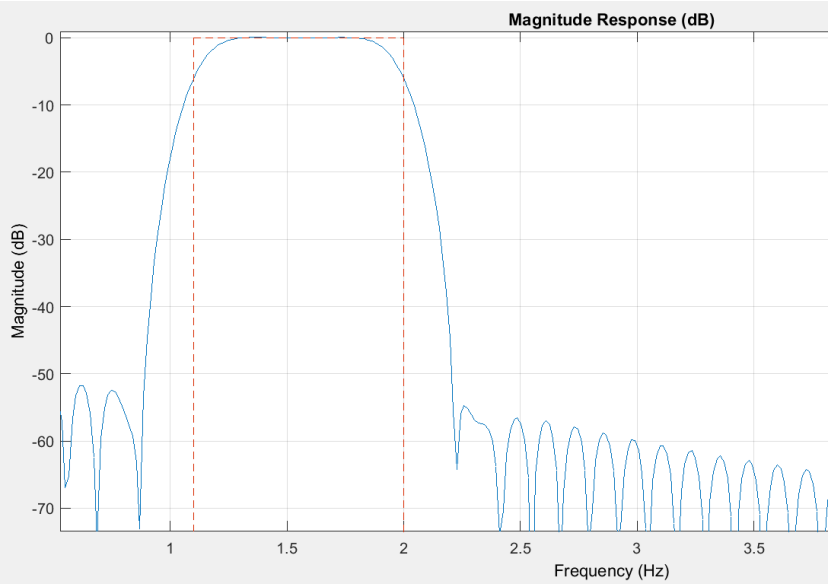


Figure 25: Algorithm 1– bandpass filter

The obtained signal after appliying the filter is the one shown in the figure 26.

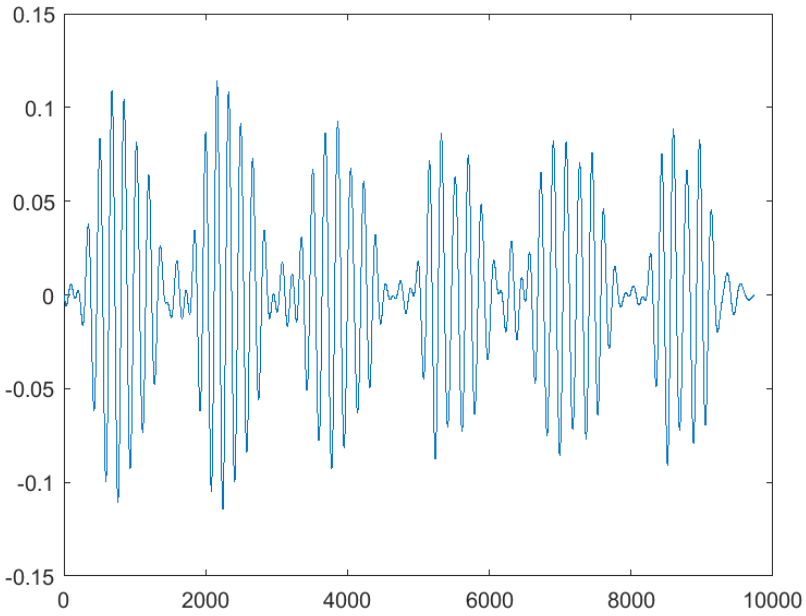


Figure 26: Algorithm 1– signal after bandpass filter

As shown in figure 26, the amplitude of the signal after filtering is very small, that is why a square function of the signal is then performed and finally normalise it.

```
% SQUARED and NORMALIZED (Ydata)
bp_Y_squared = bp_Y.^2;
bp_Y_squared = (bp_Y_squared - mean(bp_Y_squared))/std(bp_Y_squared);
bp_Y_squared = normalize(bp_Y_squared,'zscore');
```

Illustration 16: Code MATLAB algorithm 1 - squared plus normalise

The result signal is the one shown below, with an amplitude much bigger allowing to operate with the signal in a better way.

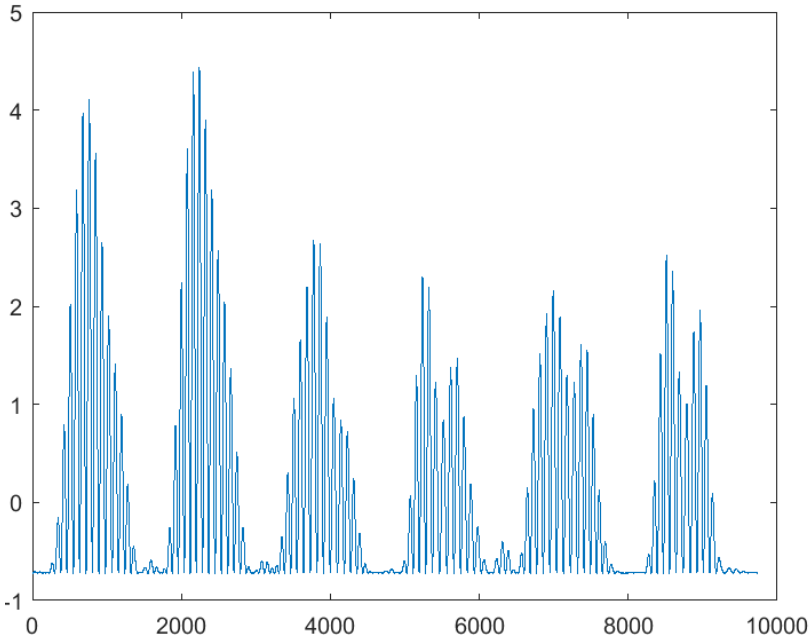


Figure 27: Algorithm 1– signal after squared plus normalized

The objective of this first algorithm, as it has been said, is to obtain, by the help of two filtering, an squared signal that collects in each block the data related of 5 steps. A second filtering is required for obtaining that. It consists of a lowpass filter. This is a filter that passes signals with a frequency lower than a selected cutoff frequency and attenuates signals with frequencies higher than the cutoff that in this case is 0,005 Hz normalized out of 250Hz.

```
%FIR lowpass filter (normalizado), dividido las frecuencias por 250Hz
f2 = designfilt('lowpassfir','PassbandFrequency',0.0001, ...
         'StopbandFrequency',0.005,'PassbandRipple',0.4, ...
         'StopbandAttenuation',65,'DesignMethod','kaiserwin');
lp_Y2 = filtfilt(f2,bp_Y_squared);

%normalize [0,1] the data already filtered
lp_Y2 = normalize(lp_Y2,'range');
```

Illustration 17: Code MATLAB algorithm 1 - lowpass filter

This second filtering (which is already normalized) is the yellow line in the figure 28.

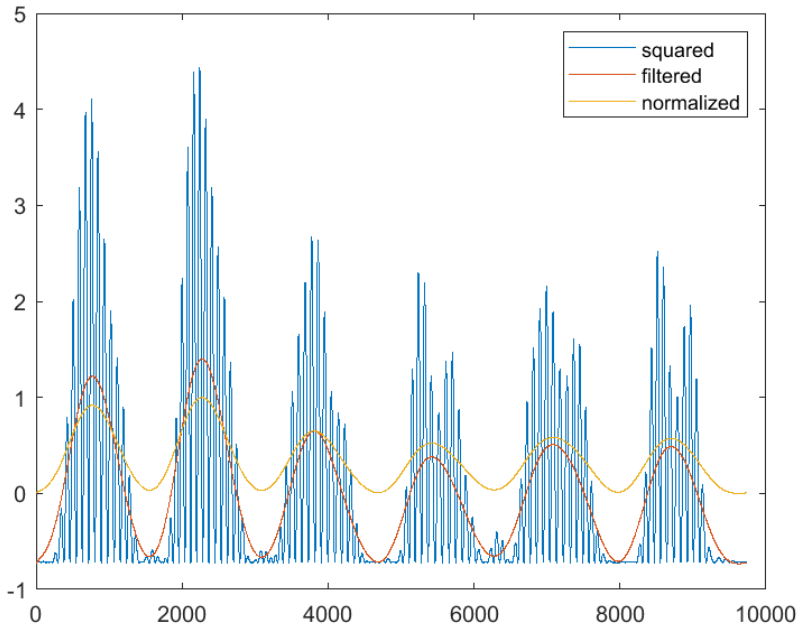


Figure 28: Algorithm 1– signal after second filter

Once it is obtained this sinusoidal signal, it is very easy to create a squared signal. The code implemented for that is not included, but the main idea is to enclose in a block every time the sinusoidal goes up and down, in this order.

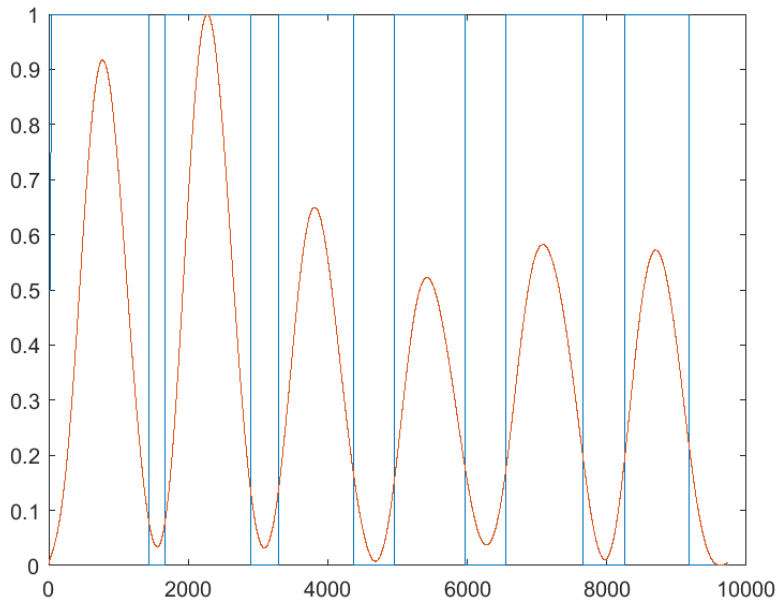


Figure 29: Algorithm 1– sinusoidal and squared signals

The next step is to create a matrix in which each column contains all the data related to a block of 5 steps, in this particular case, the matrix is formed by 6 columns.

```
% create matrix, each column with data of each positive signals
matrixPackage=zeros(length(Y),length(startIndex));
for i=1:length(startIndex)
    matrixPackage(1:endIndex(i)-startIndex(i)+1,i)=Y(startIndex(i):endIndex(i));
end
```

Illustration 18: Code MATLAB algorithm 1 - matrix of blocks

This matrix is declared with a size of 9750x6, however, each column occupy approximately 1/6 of the length.

matrixPackage

9750x6 double

	1	2	3	4	5	6
1	0.4579	0.4385	0.4369	0.4682	0.3963	0.4674
2	0.4579	0.4385	0.4396	0.4671	0.3926	0.4818
3	0.4580	0.4386	0.4431	0.4651	0.3888	0.4888
4	0.4583	0.4376	0.4480	0.4641	0.3877	0.4981
5	0.4588	0.4340	0.4528	0.4630	0.3882	0.5082
6	0.4593	0.4428	0.4573	0.4614	0.3870	0.5124
7	0.4594	0.4417	0.4607	0.4590	0.3874	0.5156
8	0.4595	0.4418	0.4626	0.4569	0.3912	0.5153
9	0.4596	0.4452	0.4642	0.4559	0.3925	0.5148
10	0.4596	0.4505	0.4652	0.4572	0.3968	0.5126

Figure 30: Algorithm 1– matrix of blocks

Finally it would only remain to find the peaks in each of these columns as shown in the code below. It has been set a minimum at 0.7, so that only peaks above this value are considered as steps.

```
%find picks, its location and width de cada columna de la matriz package,
location=zeros(length(Y),length(startIndex));
location_steps=zeros(1,length(startIndex));
for z=1:length(startIndex)
[a,c] = findpeaks(matrixPackage(:,z), 'MinPeakHeight',0.7);
c = c + startIndex(z);
location_steps(1:length(c),z)=c(:,1);
end
```

Illustration 19: Code MATLAB algorithm 1 - findpeaks

The peaks are kept in a vector, as shown in figure 31. These numbers are the x-position of the peaks in samples.

matrixPackage | location | location_steps

1x23 double

	1	2	3	4	5	6	7	8	9	10
1	508	513	695	856	869	1220	2000	2159	2330	2339

Figure 31: Algorithm 1 - matrix of peaks

With all these steps followed so far, the first algorithm model will be completed. Figure 32 represents how this model looks.

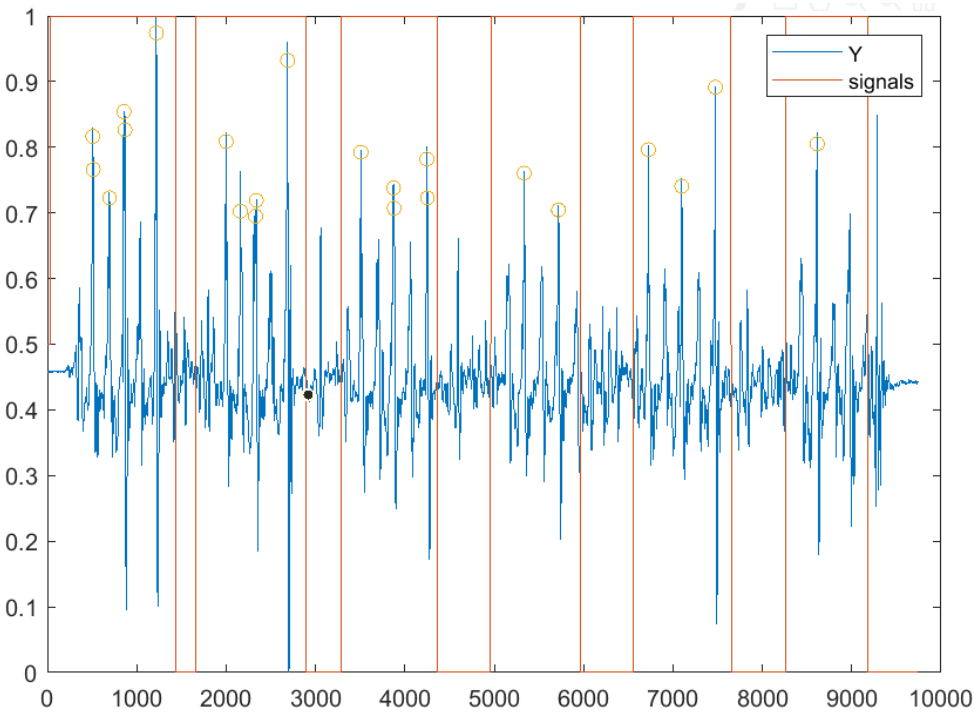


Figure 32: Algorithm 1– final model

The expected results when applying this algorithm are those shown in Figure 32, formed by the input signal (normalised) in blue colour, the square wave that collects in blocks every 5 steps in orange colour and in a yellow circle each of the peaks (steps) detected by the algorithm.

From this algorithm it is expected to obtain results like that of figure 32, in which it is able to detect every 5 steps a block and within each block the steps given.

5.2 Algorithm model 2

In this second model the square signal is not performed to the block of 5 steps but is done to each of the steps separately. This is why it only requires one filter. The steps follow until the final model are described below.

This second model starts in the same way as the first one, normalizing the input signal with the same function *normalize* between the range of 0 and 1, and realizing in addition, the cube of the signal.

```
Y1=normalize(Y,'range');
Y=Y1.^3;
```

Illustration 20: Code MATLAB algorithm 2– normalize and cubic the input signal

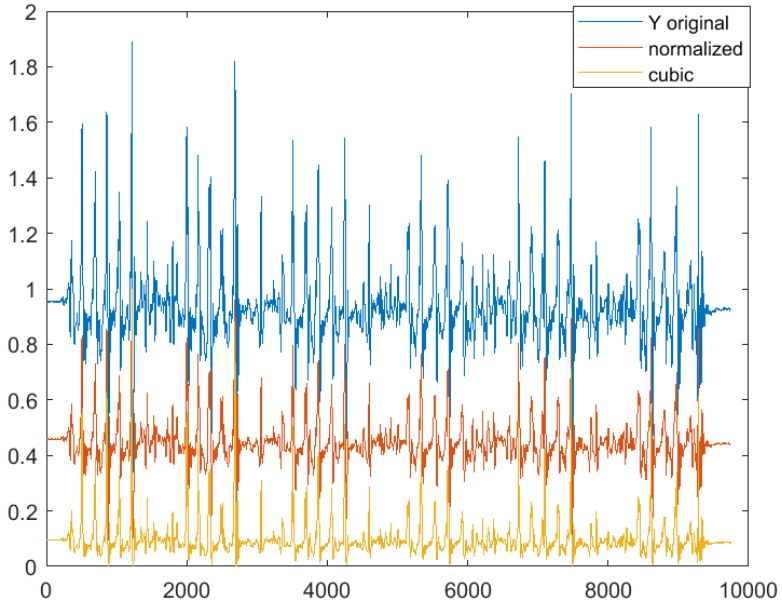


Figure 33: Algorithm 2– input signal

The next step is a Bandpass filter carried out in the same way as the one in the previous model, simply changing the frequencies to 0.5 Hz and 2.4 Hz, as seen in the illustration 21. This frequency adjustment has been done simply because empirical tests have shown that these values fit better to the desired filtering.

```
%FIR bandpass filter, between 0.5 y 2.4 Hz
bpFilt = designfilt('bandpassfir',...
         'FilterOrder',2000, ...
         'CutoffFrequency1',0.5,...
         'CutoffFrequency2',2.4, ...
         'SampleRate',250);
bp_Y21 = filtfilt(bpFilt,Y);
bp_Y2 = normalize(bp_Y21,'range');
```

Illustration 21: Code MATLAB algorithm 2– bandpass filter

The result obtained after filtering is the one shown in figure 34: comparing the normal one with the filtered.

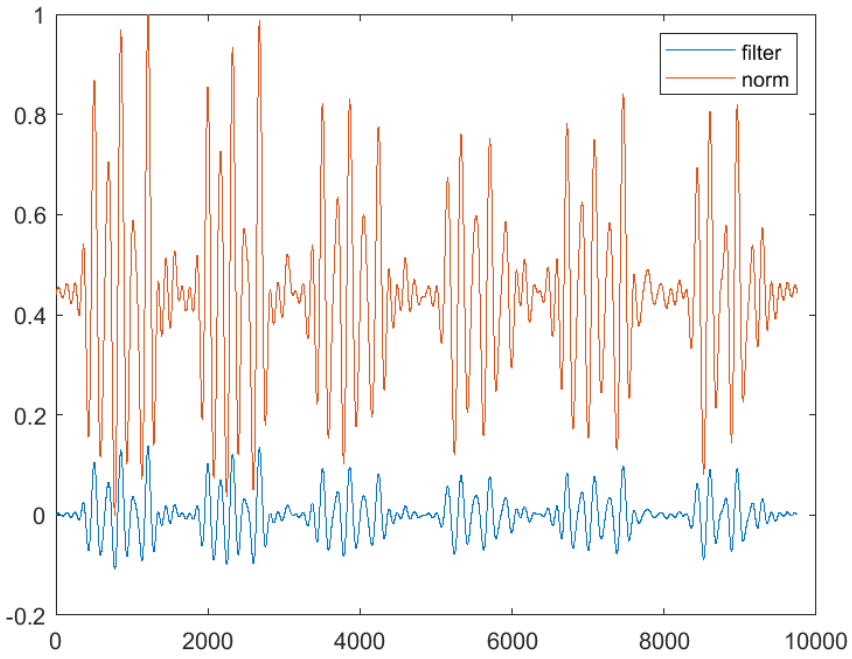


Figure 34: Algorithm 2– signal after bandpass filter

The square signal is now created. In this case it does not form a block grouping together a set of steps, but collects in each, a wave of the filtered signal. It is for this reason that the square signal is much larger than that obtained in model 1.

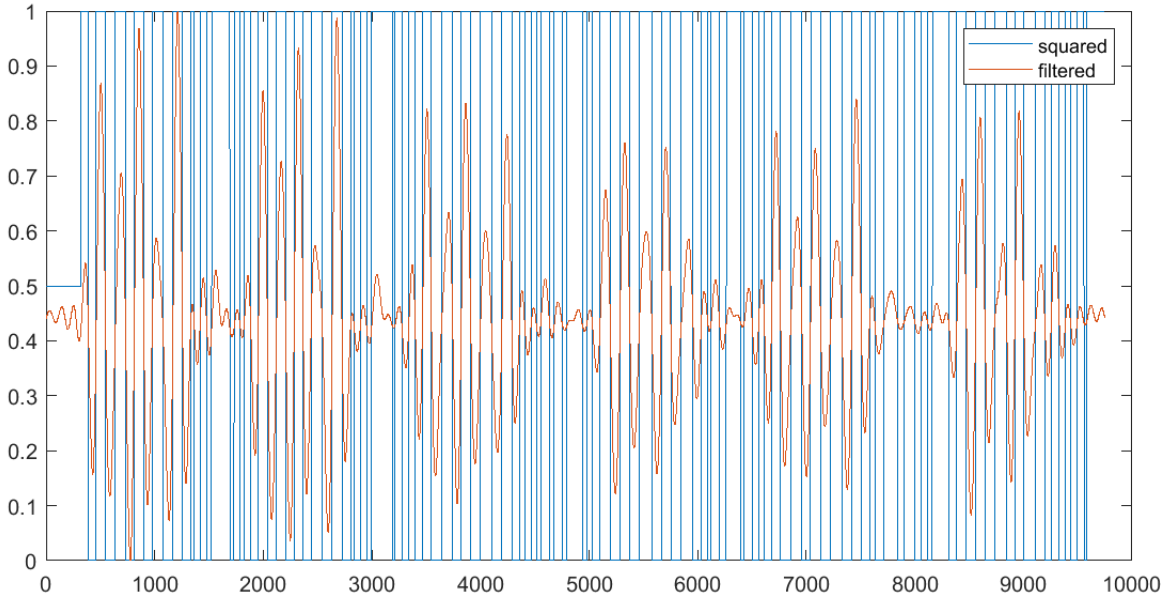


Figure 35: Algorithm 2– squared signal

In this way, the matrix is created. Each column represents each of the square waves of the squared signal. There are more columns than steps taken because it also considers the possible perturbations during the turn.

```
% create matrix, each column with data of each positive signals
matrixPackage_2=zeros(length(Y),length(startIndex_2));
for i=1:length(startIndex_2)
    matrixPackage_2(1:endIndex_2(i)-startIndex_2(i)+1,i)=bp_Y2(startIndex_2(i):endIndex_2(i));
end
```

Illustration 22: Code MATLAB algorithm 2– matrix of steps

In this example, which has been said that the file is made of a recording of 30 steps, it contains 60 columns.

9750x60 double

	1	2	3	4	5	6
1	0.4236	0.4300	0.4241	0.4285	0.4295	0.4368
2	0.4265	0.4447	0.4319	0.4476	0.4384	0.4550
3	0.4295	0.4595	0.4397	0.4667	0.4470	0.4734
4	0.4326	0.4745	0.4475	0.4859	0.4555	0.4919
5	0.4358	0.4895	0.4552	0.5051	0.4637	0.5106
6	0.4392	0.5046	0.4629	0.5243	0.4717	0.5293
7	0.4426	0.5197	0.4706	0.5434	0.4795	0.5481
8	0.4462	0.5348	0.4782	0.5625	0.4870	0.5669
9	0.4498	0.5498	0.4857	0.5814	0.4943	0.5857
10	0.4536	0.5649	0.4932	0.6002	0.5013	0.6045

Figure 36: Algorithm 2– matrix of steps

Finally, as it has been done in the algorithm above, the peaks are searched in each column. In the previous case, for each column (each block of steps) it was expected to find at least 5 steps. However, in this case, since the square wave is much more restricted, a single peak is sought.

```
%find picks, its location and width de cada columna de la matriz package
location_steps_2=zeros(1,length(startIndex_2));
for z=1:length(startIndex_2)
[a2,c2] = findpeaks(matrixPackage_2(:,z), 'MinPeakHeight',0.65,'Npeaks',1);
c2 = c2 + startIndex_2(z);
location_steps_2(1:length(c2),z)=c2(:,1);
end
```

Illustration 23: Code MATLAB algorithm 2 - findpeaks

As it was expected, the number of peaks is 30, the total of steps taken during the test.

location_steps_2

1x30 double

	1	2	3	4	5	6	7	8	9	10
1	505	693	856	1014	1213	1999	2167	2326	2476	2679

Figure 37: Algorithm 2– location peaks

Finally the results of this second model are shown in figure 38.

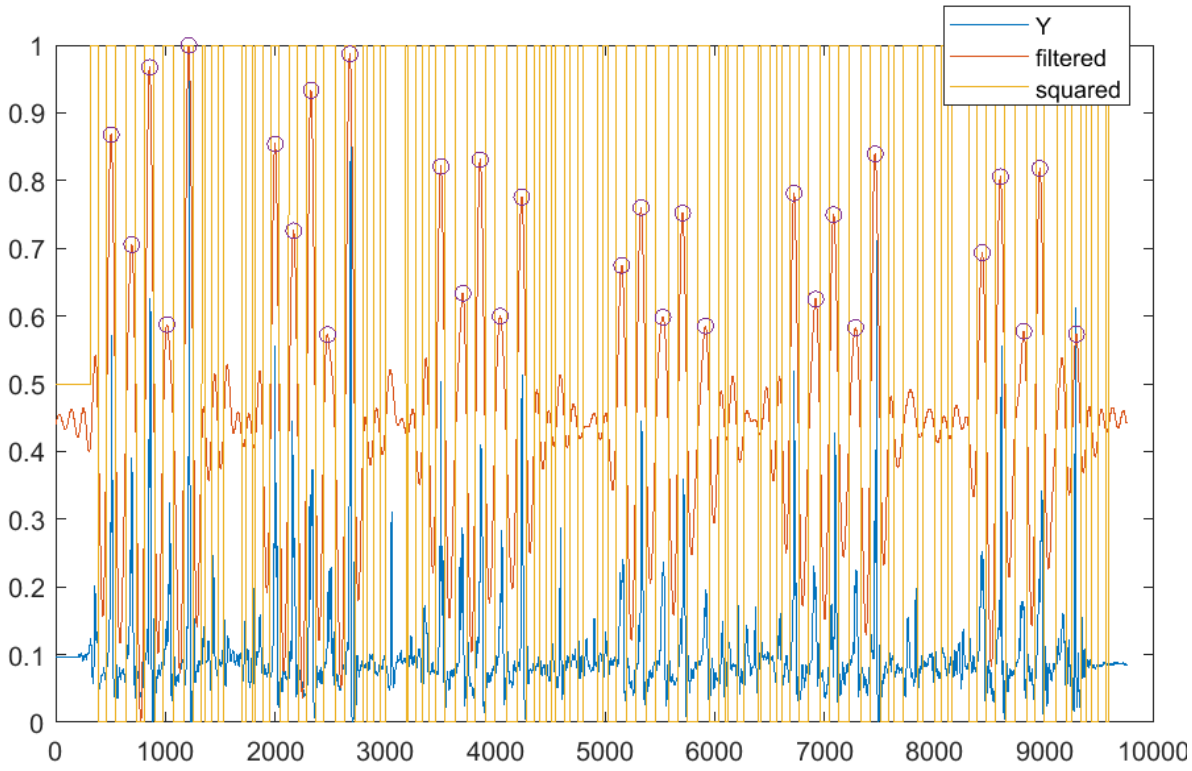


Figure 38: Algorithm 2– final model

The expected results when applying this second algorithm are those shown in Figure 38, formed by the input signal (filtered) in red colour, the square wave that collects in blocks every 5 steps in orange colour and in a black circle each of the peaks (steps) detected by the algorithm.

From this algorithm it is expected to obtain results like that of figure 38, in which it is able to detect a step every wave of the signal higher than a certain value set in 0.65.

For both algorithms (figure 32 and figure 38), the y-axe represents samples and the x-axe represents the standard input signal normalised between 0 and 1.

6 TESTS AND RESULTS

Once the models have been perfectly defined, a series of tests have been taken by two different testers:

- Test 1: walking tests at a slow speed.
- Test 2: walking tests at a normal speed.
- Test 3: walking tests at a fast speed.

Test have been carried out with time not with number of steps like the one used for setting the algorithms. The specific time set for all the tests is 35 times de 3,9 seconds stablished for acquiring 15 packets (illustration 9). This is why the number of steps at each test (inside the same category) are never the same. Both testers have tried to get the same number in both tests, but several factors influences in this: the way of walking, the breadth of the strides, the way the testers spin, etc.

TESTER 1		
File .h	**Test**	**steps**
6	slow	50
7	slow	53
9	normal	95
12	normal	110
11	fast	124
13	fast	130

Table 4: Experiments taken by tester 1

TESTER 2		
File .h	**Test**	**steps**
14	slow	47
15	slow	48
16	normal	85
17	normal	100
21	fast	129
22	fast	130

Table 3: Experiments taken by tester 2

It has been tried to reach in the slow mode about 50 steps, in the normal mode over 100 steps and in the fast one approximately 130 steps.

A total of 24 tests have been carried out, two for each tester and for each test. In the next paragraph, half of the results have been included, the other half have been attached in the annexes. A total of 4 tests for each type are shown, 2 for each algorithm and for each tester.

All the graphs obtained will be shown below and in the results section all will be commented together.

6.1 Test 1

Type 1 tests consist of recording 50 steps at a slow speed. This means, in the programmed time, about 5 turns, considering a turn one way and back (10 steps).

The results of the first algorithm at slow speed are:

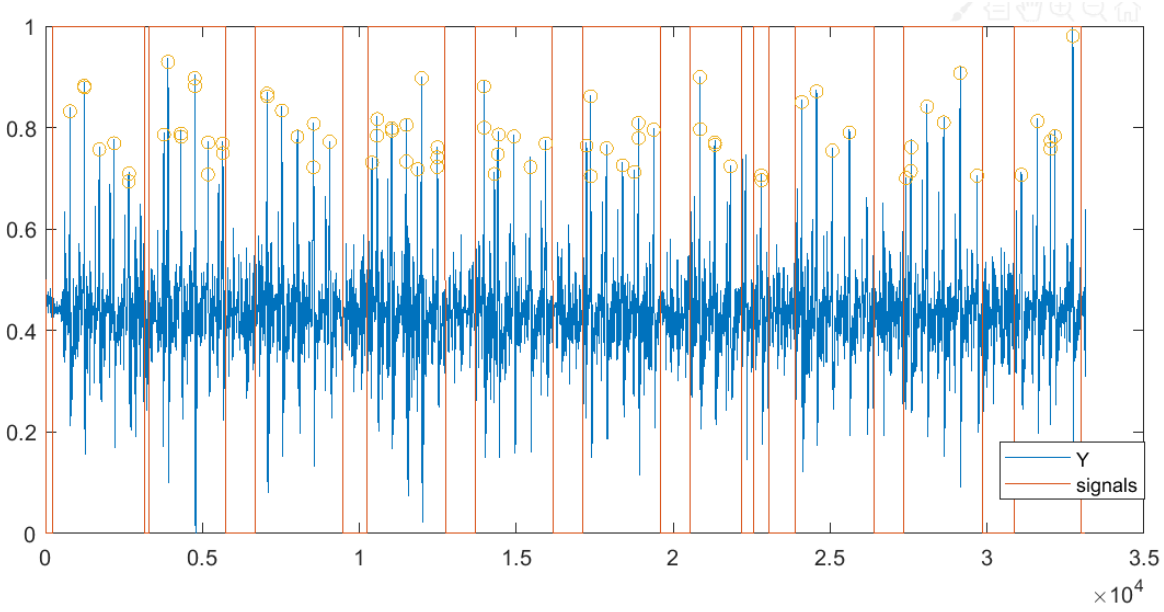


Figure 39: Algorithm 1 – tester 1

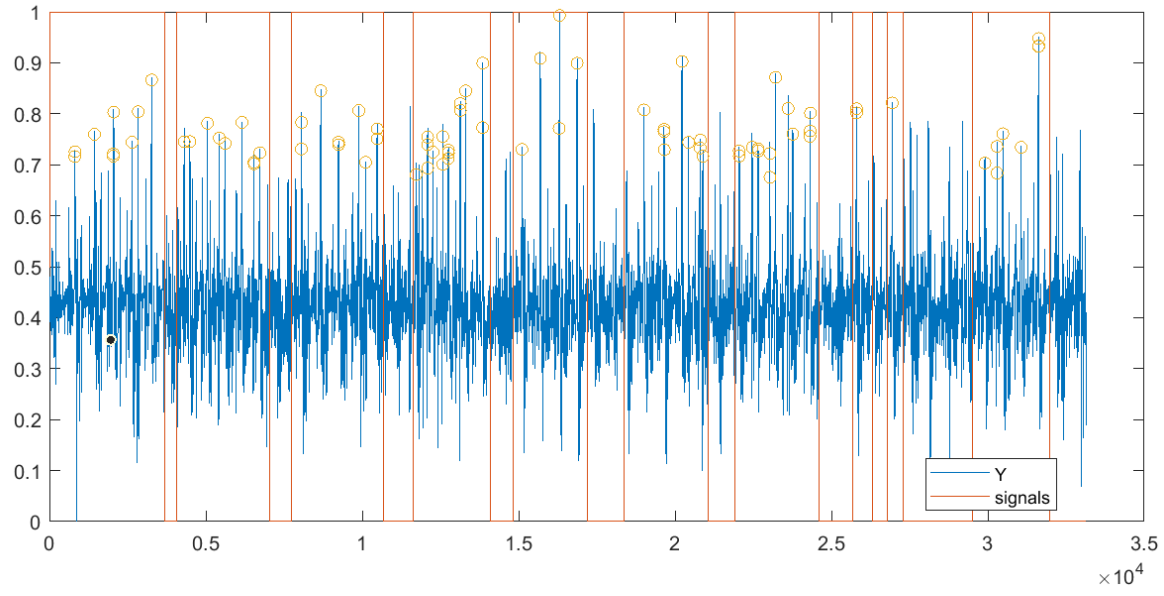


Figure 40: Algorithm 1 – tester 2

As it can be seen in this two grafics from the algorithm 1, as it was expected, there are over 10 blocks of the square wave in each, in which more than 5 steps are collected.

The results of the second algorithm at slow speed are:

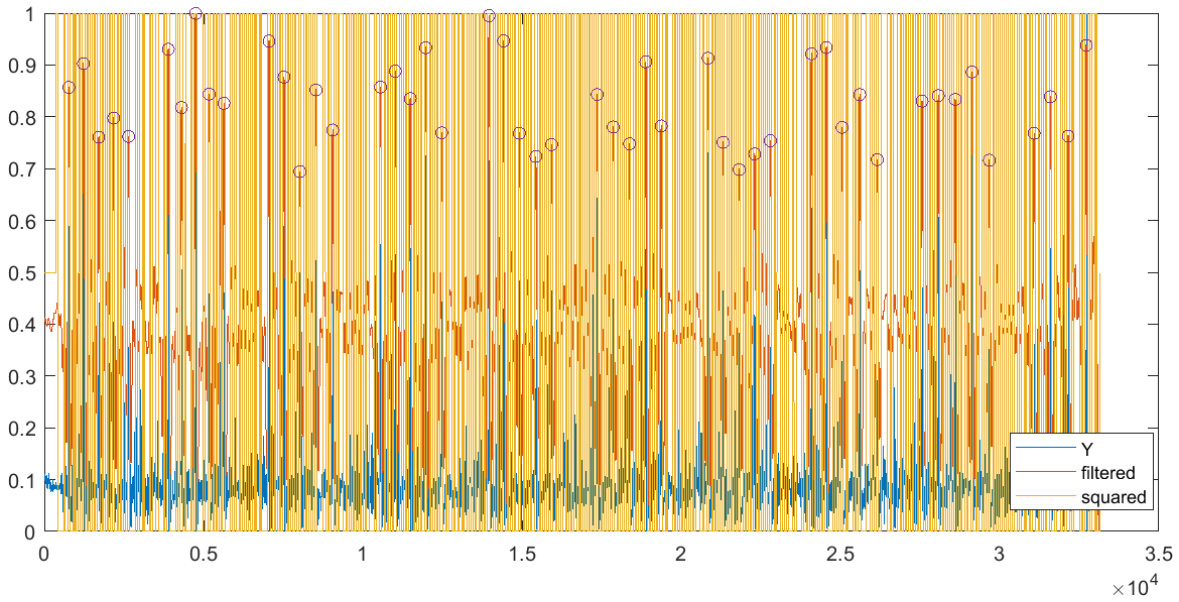


Figure 41: Algorithm 2 – tester 1

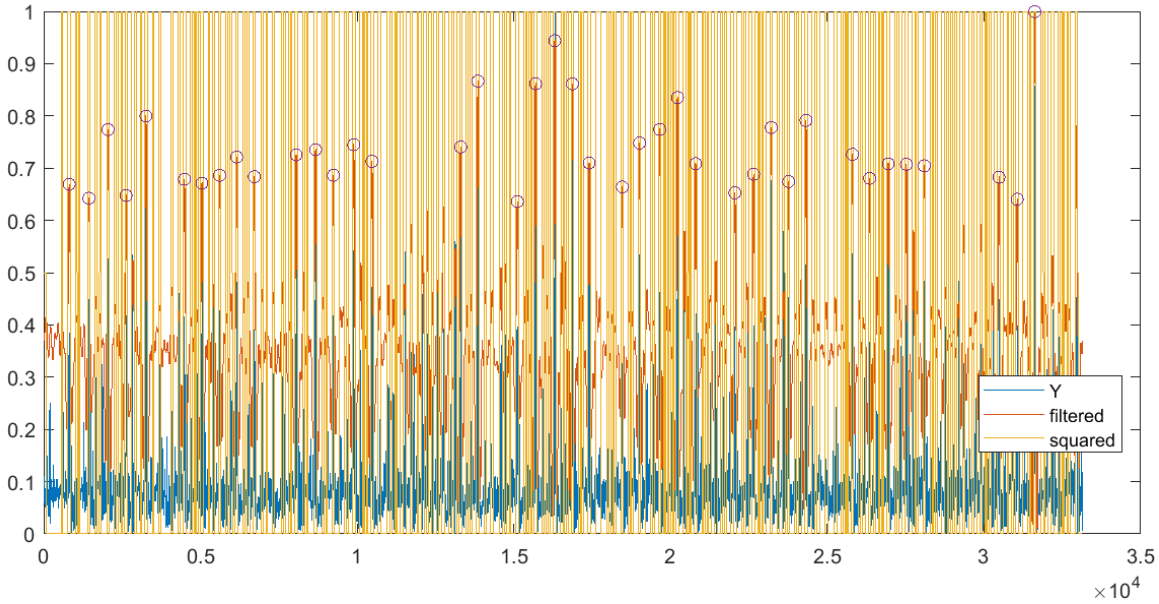


Figure 42: Algorithm 2 – tester 2

In this case the results of both testers are similar. By setting the limit at the 0.65 commented, 5 steps are detected at each turn.

6.2 Test 2

Type 2 tests consist of recording 100 steps at a normal speed. This means, in the programmed time, about 10 turns, considering a turn one way and back (10 steps).

The results of the first algorithm at normal speed are:

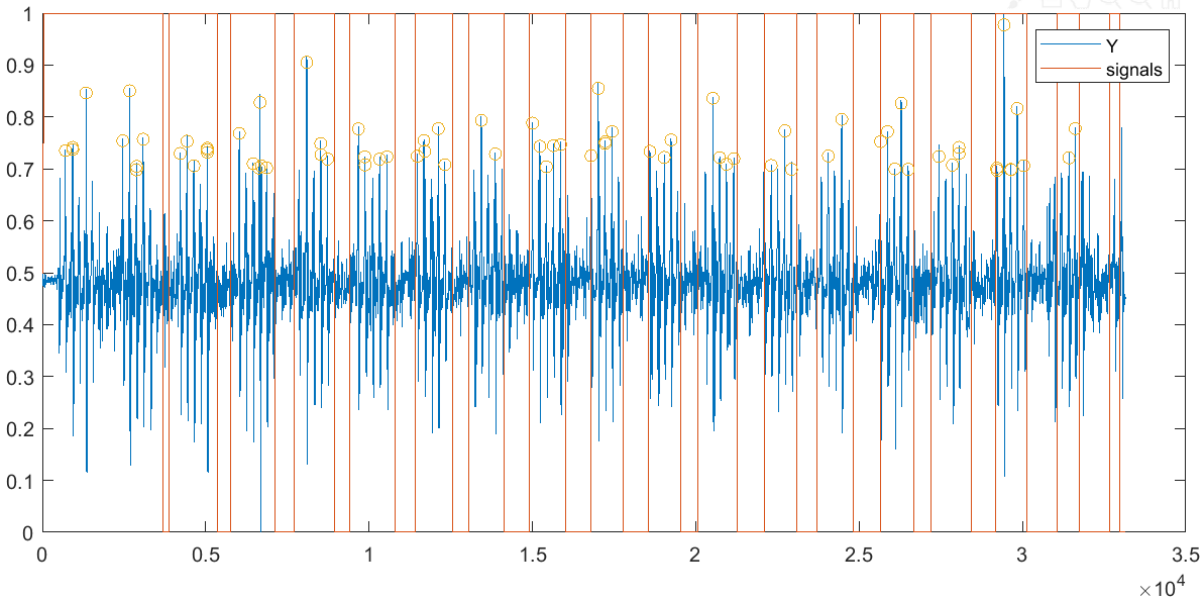


Figure 43: Algorithm 1 – tester 1

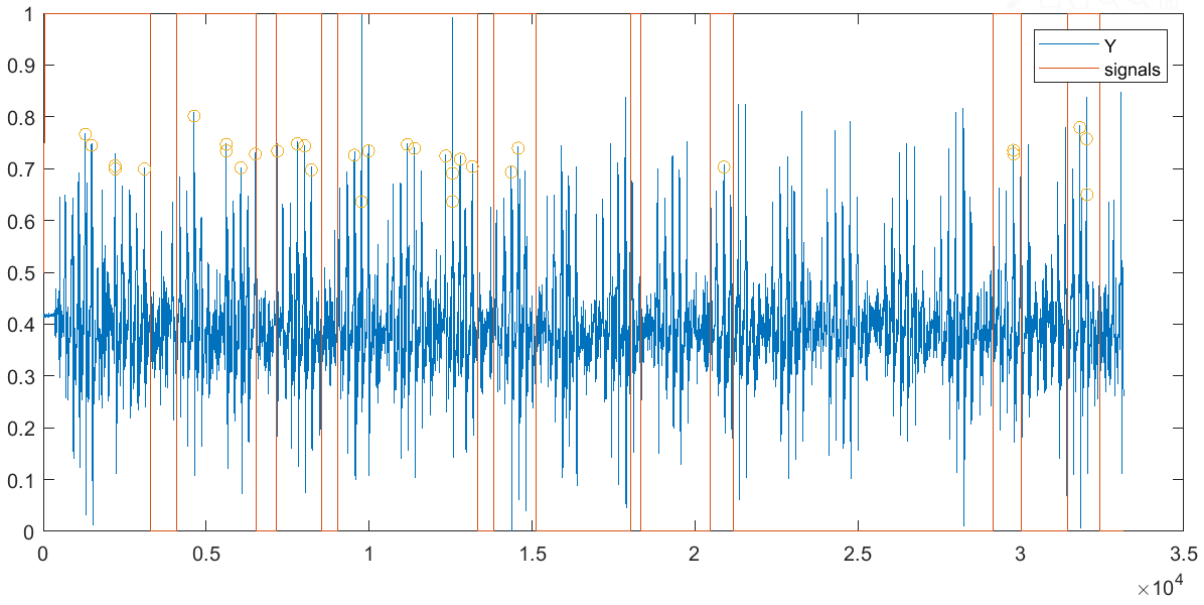


Figure 44: Algorithm 1 – tester 2

It is noted that when the speed is increases, the algorithm is not as effective due to it does not stablish clearly the blocks of 5 steps. It was supposed to detect 20 blocks of the squared form, however, neither of them gets this.

The results of the second algorithm at normal speed are:

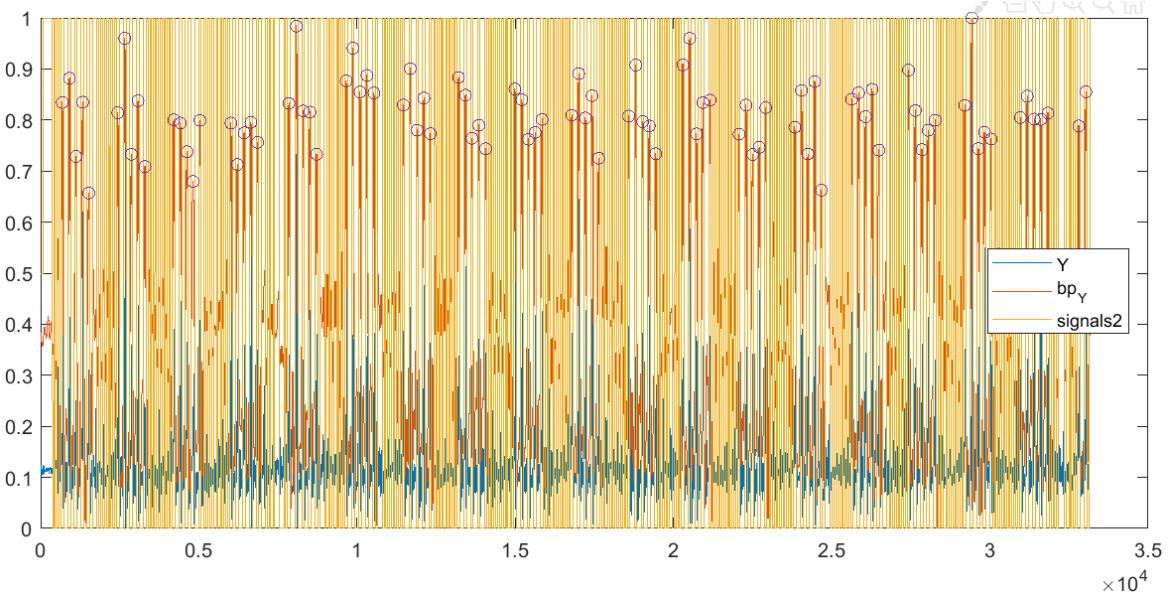


Figure 45: Algorithm 2 – tester 1

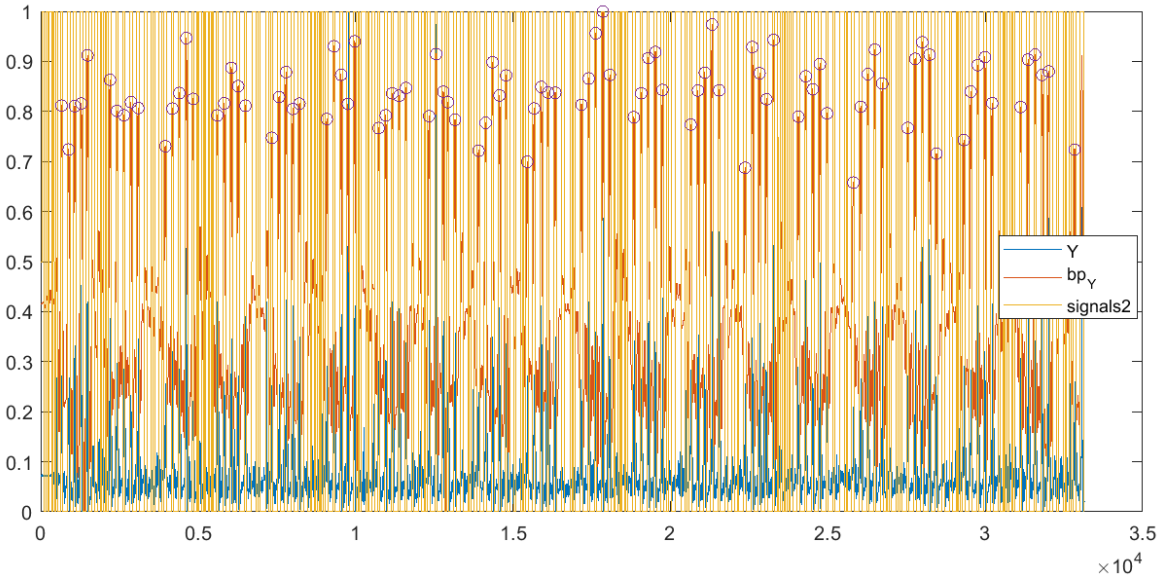


Figure 46: Algorithm 2 – tester 2

However, with algorithm 2 it occurs the opposite. Whenever the speed is increased, it gets a greater precision and even greater stability. In this case again the results of both testers are similar. Tall the steps are detected perfectly.

6.3 Test 3

Type 3 tests consist of recording 130 steps at a fast speed. This means, in the programmed time, about 13 turns, considering a turn one way and back (10 steps).

The results of the first algorithm at fast speed are:

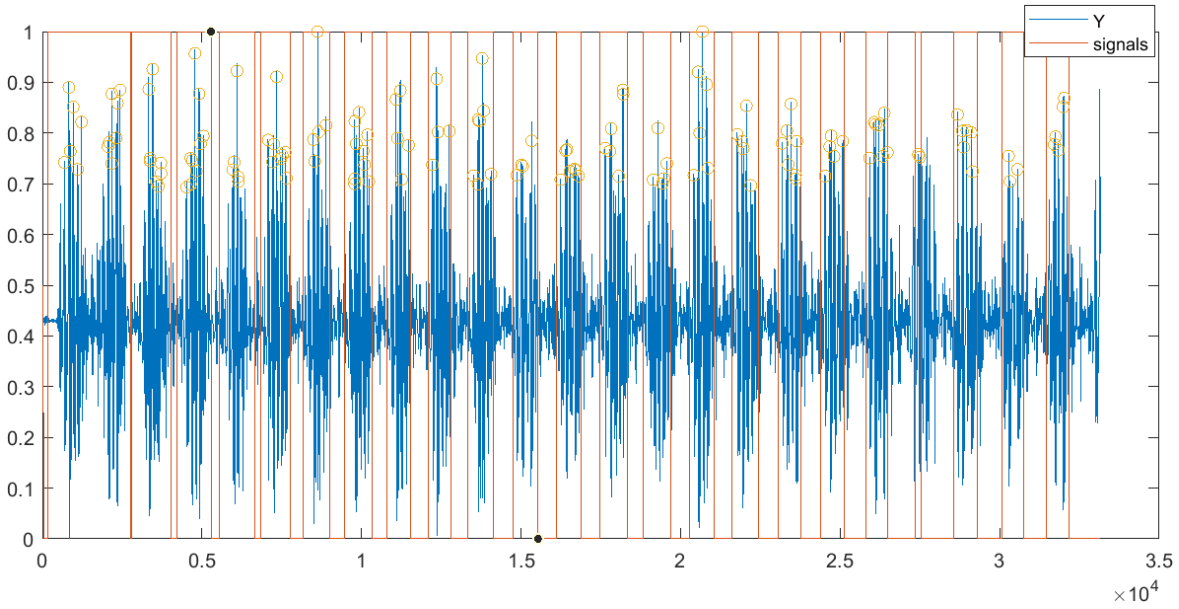


Figure 47: Algorithm 1 – tester 1

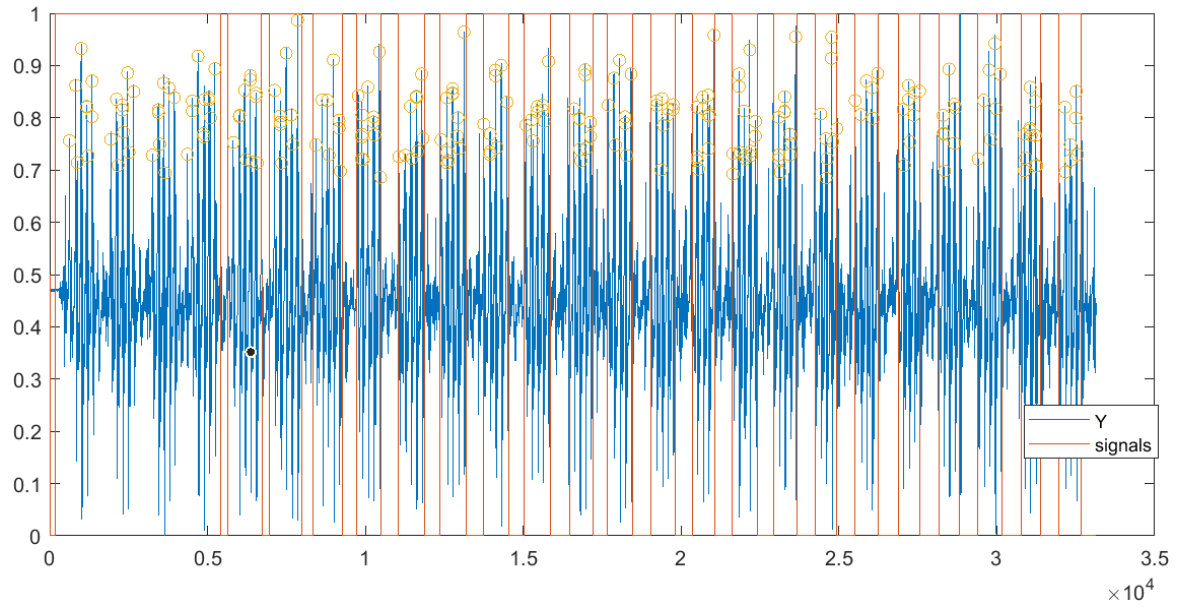


Figure 48: Algorithm 1 – tester 2

It confirms the commented above, at higher speed the precision is lower.

The results of the second algorithm at fast speed are:

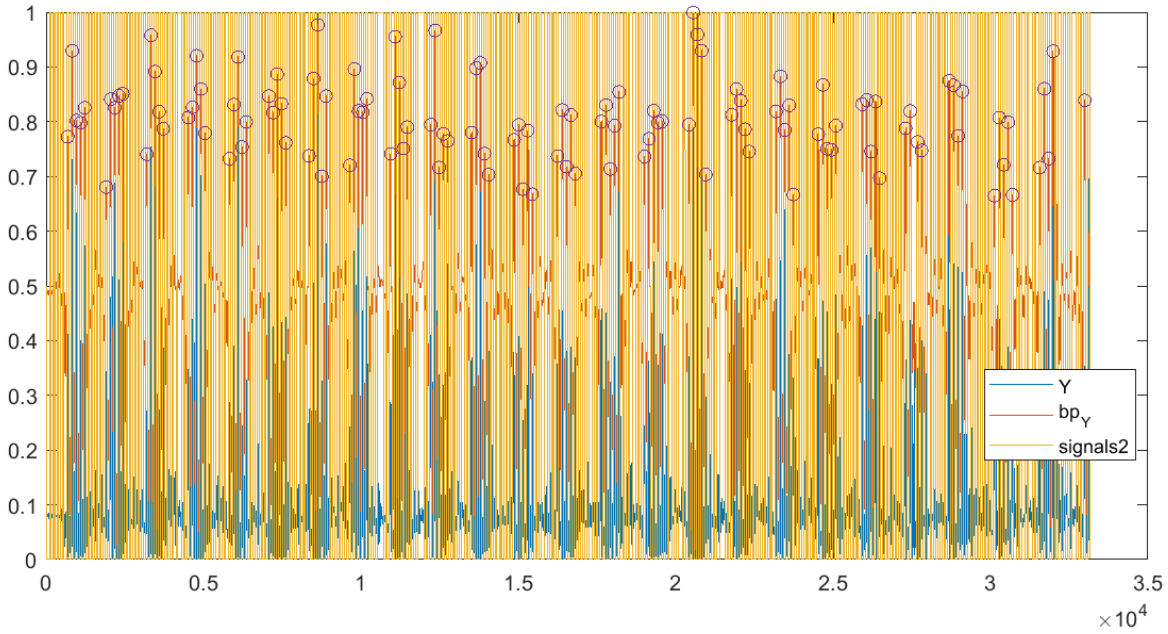


Figure 49: Algorithm 2 – tester 1

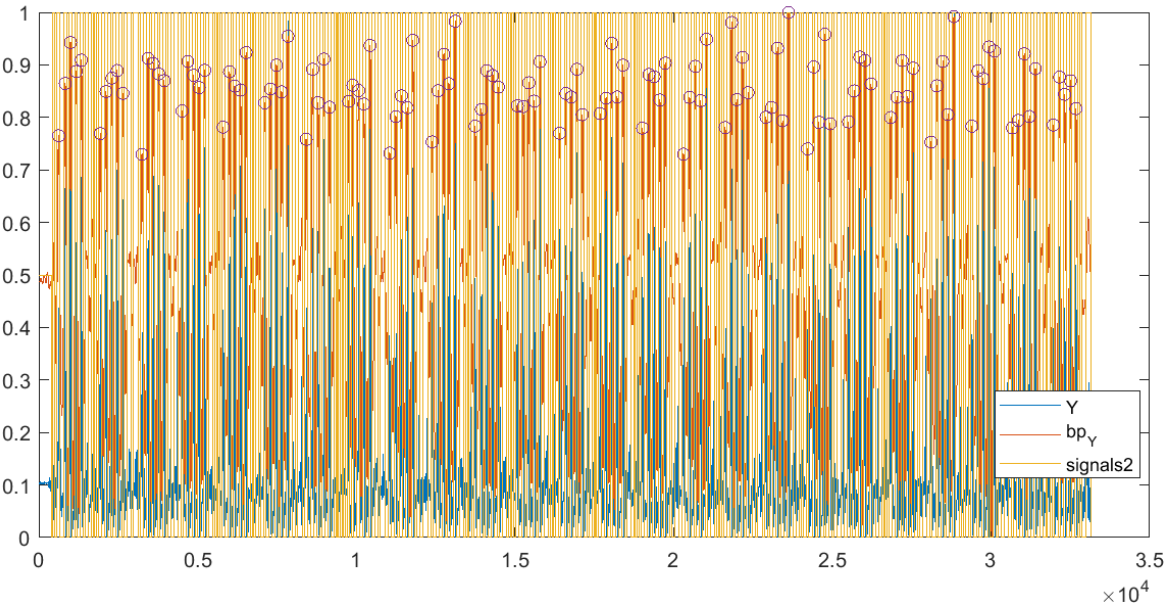


Figure 50: Algorithm 2 – tester 2

In this last case, it is confirmed too that speed provides higher precision in the algorithm. All steps are detected perfectly in both testers.

6.4 Interpretation of the results

We have just shown that our system, hardware and software is working as predicted when making the tests. The results will be commented in function of the algorithm used. So first the viability of algorithm 1 and then that of algorithm 2.

ALGORITHM 1

The following conclusions can be drawn from figures 39, 40, 43, 44, 47 and 48:

This type of algorithm as said previously is limited to find the peaks within each of the blocks formed by 5 steps. In order to do this, a limit is set, so that all peaks above this value (set in 0.7) are considered as steps. This leads to a huge error, as it depends a lot on the signal filtering. Although every attempt has been made to obtain the filtering as accurate as possible, as shown in the figures, there is a fairly high error. In all the blocks there are more than 5 peaks.

It can be checked that in the cases where the tester has a higher speed, the probability of the algorithm of finding the block of the 5 steps is lower. This is certainly because the rotation transition is done very quickly, not giving enough time to the algorithm to detect it as a turn. This can be seen in figures 44 and 48 at both normal and fast speeds.

Another aspect that has been observed, is that the quicker the tests are carried out, the more chaotic behaviour the figures show. This means that the algorithm is not sensitive enough for speed changes, which is assumed to be a very negative aspect.

ALGORITHM 2

The following conclusions can be drawn from figures 41, 42, 45, 46, 49 and 50:

This algorithm, compared to the previous one, is limited to find 1 peak for each part of the signal that has been collected within the square wave. This gives it much more precision, as there is only one chance of finding one peak. In this case, it is established a limit too from which all the peaks above are considered steps, in this algorithm this value is 0,65. As can be seen in the figures, this algorithm is much more accurate and less complicated in terms of finding the steps above this value, instead, if this peak does not exceed the limit value, the goal of counting steps is lost.

One characteristic that is observed in the obtained results is that the higher the speed of the tests, the greater the stability of the peaks is achieved. That is, if it is looked at the peaks found in all the figures, in the tests of normal and fast type the peaks are less dispersed than in the tests of type slow.

This can also be compared with the results obtained in the algorithm 1, where the dispersion is much greater and remarkable. This is only due to the type of filtration that has been carried out in both algorithms.

7 CONCLUSIONS AND FUTURE WORKS

In this paper, it has been demonstrated the feasibility of using a wireless sensor network to design two models that will be able to detect fall events.

The sensor used in the measurements was assembled and programmed in the scope of this work. It features an accelerometer as the main detection instrument, a programmable microcontroller as a control system and an hdf5 file in MATLAB to store the measured data. The sensor is able to operate independently of an external power supply. The final system is based on a belt that incorporates all the devices mentioned.

In this thesis, two different methods have been implemented to detect the steps of a person while walking, wearing the implemented belt. With the results obtained after testing by two testers, it has been found that both have some advantages and disadvantages.

On the one hand, the first algorithm provides a greater probability of finding all steps carried out during the test. The square wave, which is the one that delimits the search area of the peaks, encompasses 5 steps, giving to this algorithm a wide possibility of finding the peaks (the steps).

On the other hand, however, being this area of searching peaks so big in the algorithm 1, makes it to be less accurate. With algorithm 2, if the search limit is well set, the accuracy of the algorithm is very high, as it has been demonstrated in all the tests carried out.

It can be conclude from the results obtained that it is possibly to perform reliable data acquisition and analysis using low-cost and easily available hardware and software for implementing a future fall detection system. It is still too early to determine if the models proposed are the most appropriate for the fall detection system, however, algorithm 2 is a valid candidate for counting steps, leading the development of a fall detection system in the future.

In future work it could be also combine these sensor with a GPS chip, to provide localization of the person outside the home. Visual environmental factors such as lighting levels and room layout changes can be significant for many elderly people with poor vision and are implicated in falls. Automatic detection of such changes can also be under investigation.

BIBLIOGRAPHIC REFERENCES

[1] https://learn.sparkfun.com/tutorials/accelerometer-basics/all

[2] http://www.pcb.com/Resources/Technical-Information/mems-accelerometers

[3] Analog Devices. *Low Noise, Low Drift, Low Power, 3-Axis MEMS AccelerometersADXL354/ADXL355*.2016. https://www.analog.com/media/en/technical-documentation/data-sheets/adxl354_355.pdf

[4] https://learn.sparkfun.com/tutorials/accelerometer-basics/all

[5] https://www.arduino.cc/en/reference/SPI

[6] https://store.arduino.cc/mkr-wifi-1010

[7] https://www.arduino.cc/en/Main/Software

[8] https://aprendiendoarduino.wordpress.com/2016/11/13/interrupciones/

[9] https://www.electronics-tutorials.ws/filter/filter_4.html

ANNEX

Table 1: Accelerometer - Digital outputs

DIGITAL OUTPUT FOR THE ADXL355

T_A = 25°C, V_{SUPPLY} = 3.3 V, x-axis acceleration and y-axis acceleration = 0 *g*, and z-axis acceleration = 1 *g*, and output data rate (ODR) = 500 Hz, unless otherwise noted. Note that multifunction pin names may be referenced by their relevant function only.

Table 2.

Parameter	Test Conditions/Comments	Min	Typ	Max	Unit
SENSOR INPUT	Each axis				
Output Full Scale Range (FSR)	User selectable		±2.048		*g*
			±4.096		*g*
			±8.192		*g*
Nonlinearity	±2 *g*		0.1		% FS
Cross Axis Sensitivity			1		%
SENSITIVITY	Each axis				
X-Axis, Y-Axis, and Z-Axis Sensitivity	±2 *g*	235,520	256,000	276,480	LSB/*g*
	±4 *g*	117,760	128,000	138,240	LSB/*g*
	±8 *g*	58,880	64,000	69,120	LSB/*g*
X-Axis, Y-Axis, and Z-Axis Scale Factor	±2 *g*		3.9		µg/LSB
	±4 *g*		7.8		µg/LSB
	±8 *g*		15.6		µg/LSB
Sensitivity Change due to Temperature	−40°C to +125°C		±0.01		%/°C
0 *g* OFFSET	Each axis, ±2 *g*				
X-Axis, Y-Axis, and Z-Axis 0 *g* Output		−75	±25	+75	mg
0 *g* Offset vs. Temperature (X-Axis, Y-Axis, and Z-Axis)[1]	−40°C to +125°C	−0.15	±0.02	+0.15	mg/°C
Repeatability[2]	X-axis and y-axis		±3.5		mg
	Z-axis		±9		mg
Vibration Rectification[3]	±2 *g* range, in a 1 *g* orientation, offset due to 2.5 *g* rms vibration		<0.4		*g*
NOISE DENSITY	±2 *g*				
X-Axis, Y-Axis, and Z-Axis			25		µg/√Hz
Velocity Random Walk	X-axis and y-axis		9		µm/sec/√Hr
	Z-axis		13		µm/sec/√Hr
OUTPUT DATA RATE AND BANDWIDTH					
Low-Pass Filter Passband Frequency	User programmable, Register 0x28	1		1000	Hz
High-Pass Filter Passband Frequency When Enabled (Disabled by Default)	User programmable, Register 0x28 for 4 kHz ODR	0.0095		10	Hz

Table 2: Accelerometer - pin configuration

Table 8. ADXL355 Pin Function Descriptions

Pin No.	Mnemonic	Description
1	$\overline{CS}$/SCL	Chip Select for SPI ($\overline{CS}$). Serial Communications Clock for I²C (SCL).
2	SCLK/V_{SSIO}	Serial Communications Clock for SPI (SCLK). Connect to V_{SSIO} for I²C (V_{SSIO}).
3	MOSI/SDA	Master Output, Slave Input for SPI (MOSI). Serial Data for I²C (SDA).
4	MISO/ASEL	Master Input, Slave Output for SPI (MISO). Alternate I²C Address Select for I²C (ASEL).
5	V_{DDIO}	Digital Interface Supply Voltage.
6	V_{SSIO}	Digital Ground.
7	RESERVED	Reserved. This pin can be connected to ground or left open.
8	V_{1P8DIG}	Digital Supply. This pin requires a decoupling capacitor. If V_{SUPPLY} connects to V_{SS}, supply the voltage to this pin externally.
9	V_{SS}	Analog Ground.
10	V_{1P8ANA}	Analog Supply. This pin requires a decoupling capacitor. If V_{SUPPLY} connects to V_{SS}, supply the voltage to this pin externally.
11	V_{SUPPLY}	Supply Voltage. When V_{SUPPLY} equals 2.25 V to 3.6 V, V_{SUPPLY} enables the internal LDOs to generate V_{1P8DIG} and V_{1P8ANA}. For V_{SUPPLY} = V_{SS}, V_{1P8DIG} and V_{1P8ANA} are externally supplied.
12	INT1	Interrupt Pin 1.
13	INT2	Interrupt Pin 2.
14	DRDY	Data Ready Pin.

Tables 3: Acceleration data registers

X-DATA

X-AXIS DATA REGISTERS

These three registers contain the x-axis acceleration data. Data is left justified and formatted as twos complement.

Address: 0x08, Reset: 0x00, Name: XDATA3

Table 23. Bit Descriptions for XDATA3

Bits	Bit Name	Settings	Description	Reset	Access
[7:0]	XDATA, Bits[19:12]		X-axis data	0x0	R

Address: 0x09, Reset: 0x00, Name: XDATA2

Table 24. Bit Descriptions for XDATA2

Bits	Bit Name	Settings	Description	Reset	Access
[7:0]	XDATA, Bits[11:4]		X-axis data	0x0	R

Address: 0x0A, Reset: 0x00, Name: XDATA1

Table 25. Bit Descriptions for XDATA1

Bits	Bit Name	Settings	Description	Reset	Access
[7:4]	XDATA, Bits[3:0]		X-axis data	0x0	R
[3:0]	Reserved		Reserved	0x0	R

Y-DATA

Y-AXIS DATA REGISTERS

These three registers contain the y-axis acceleration data. Data is left justified and formatted as twos complement.

Address: 0x0B, Reset: 0x00, Name: YDATA3

Table 26. Bit Descriptions for YDATA3

Bits	Bit Name	Settings	Description	Reset	Access
[7:0]	YDATA, Bits[19:12]		Y-axis data	0x0	R

Address: 0x0C, Reset: 0x00, Name: YDATA2

Table 27. Bit Descriptions for YDATA2

Bits	Bit Name	Settings	Description	Reset	Access
[7:0]	YDATA, Bits[11:4]		Y-axis data	0x0	R

Address: 0x0D, Reset: 0x00, Name: YDATA1

Table 28. Bit Descriptions for YDATA1

Bits	Bit Name	Settings	Description	Reset	Access
[7:4]	YDATA, Bits[3:0]		Y-axis data	0x0	R
[3:0]	Reserved		Reserved	0x0	R

Z-DATA

Z-AXIS DATA REGISTERS

These three registers contain the z-axis acceleration data. Data is left justified and formatted as twos complement.

Address: 0x0E, Reset: 0x00, Name: ZDATA3

Table 29. Bit Descriptions for ZDATA3

Bits	Bit Name	Settings	Description	Reset	Access
[7:0]	ZDATA, Bits[19:12]		Z-axis data	0x0	R

Address: 0x0F, Reset: 0x00, Name: ZDATA2

Table 30. Bit Descriptions for ZDATA2

Bits	Bit Name	Settings	Description	Reset	Access
[7:0]	ZDATA, Bits[11:4]		Z-axis data	0x0	R

Address: 0x10, Reset: 0x00, Name: ZDATA1

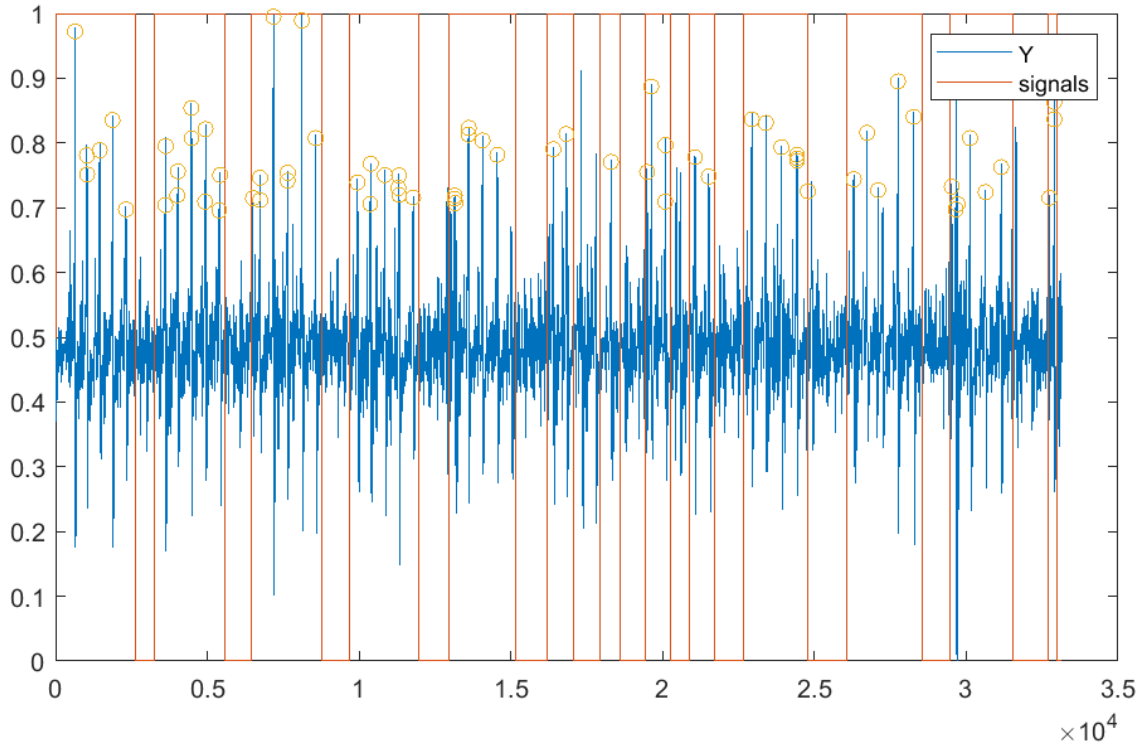
Table 31. Bit Descriptions for ZDATA1

Bits	Bit Name	Settings	Description	Reset	Access
[7:4]	ZDATA, Bits[3:0]		Z-axis data	0x0	R
[3:0]	Reserved		Reserved	0x0	R

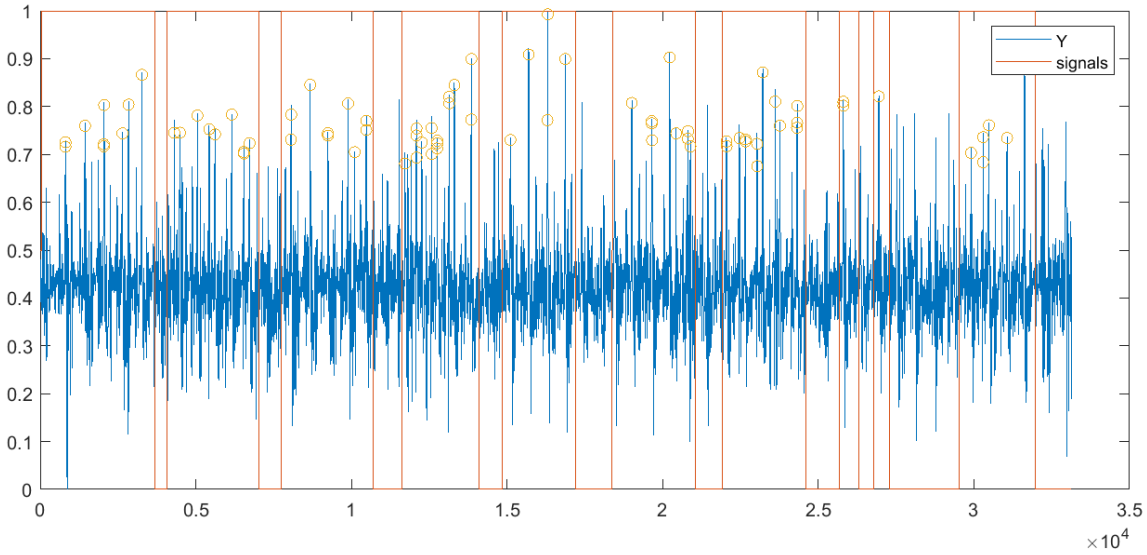
SECOND TESTS

TEST 1

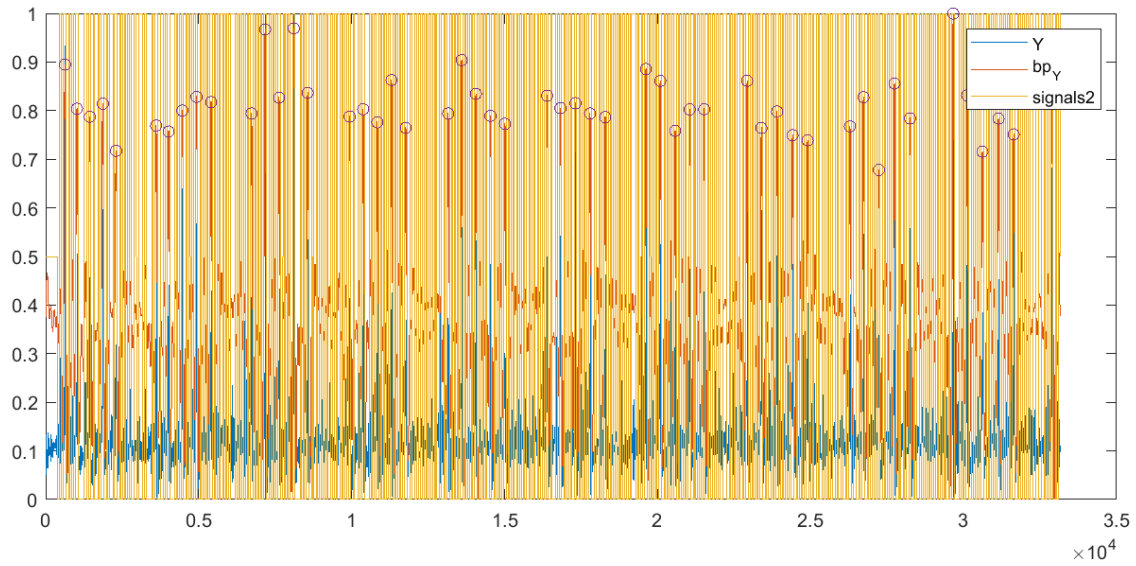
Algorithm 1 – tester 1:



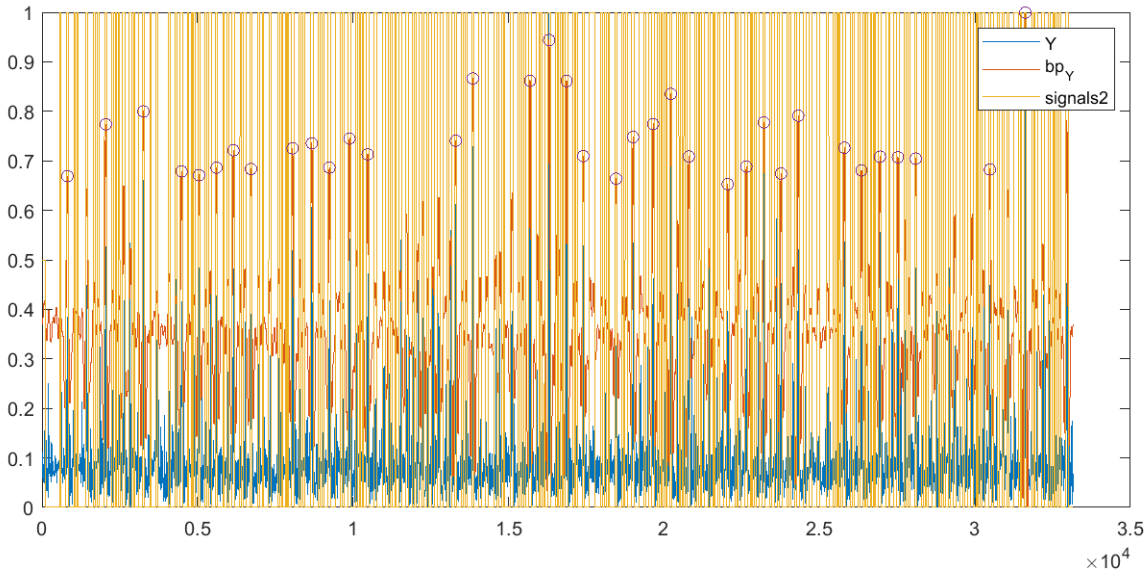
Algorithm 1 – tester 2:



Algorithm 2 – tester 1:

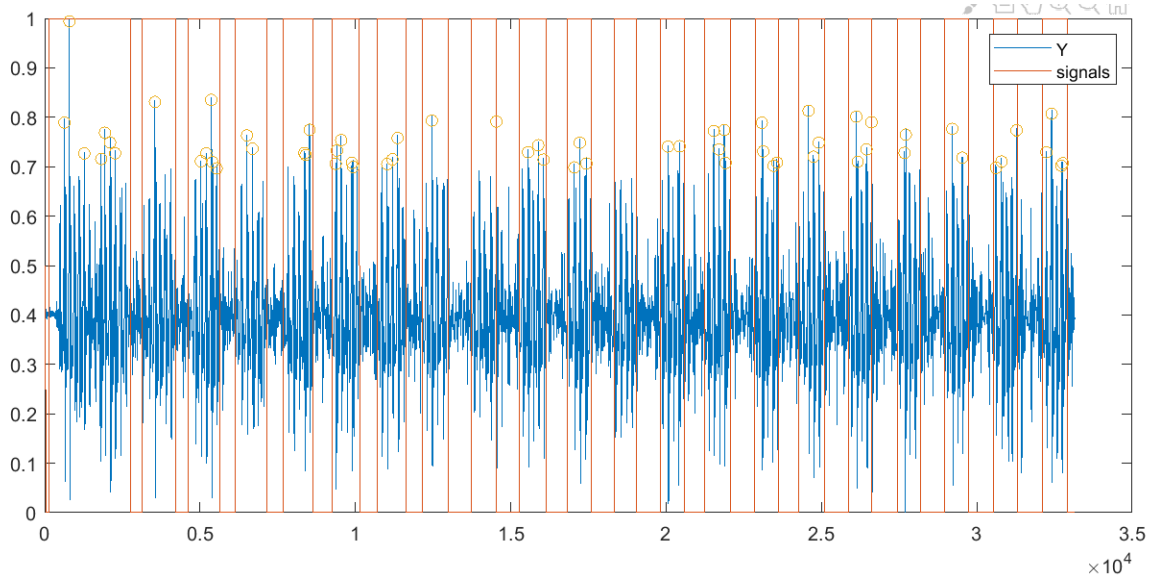


Algorithm 2 – tester 2:

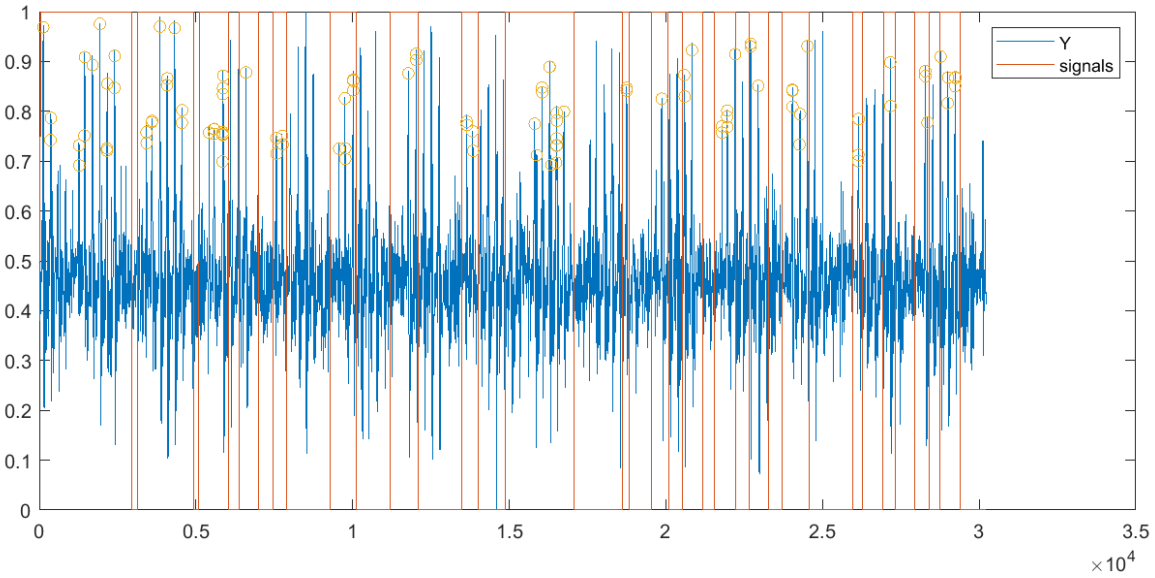


TEST 2

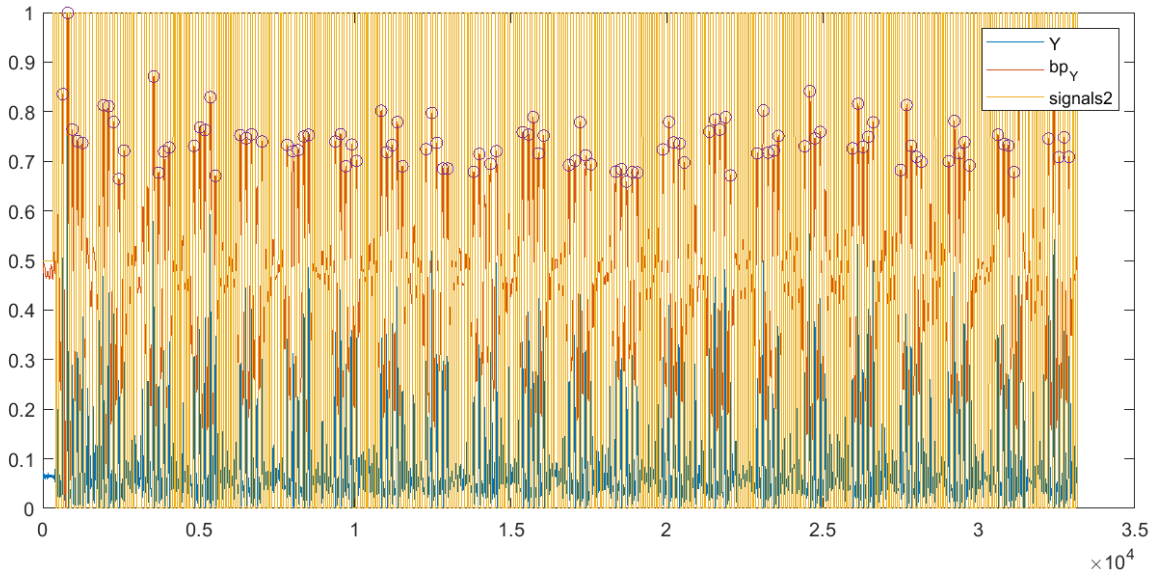
Algorithm 1 – tester 1:



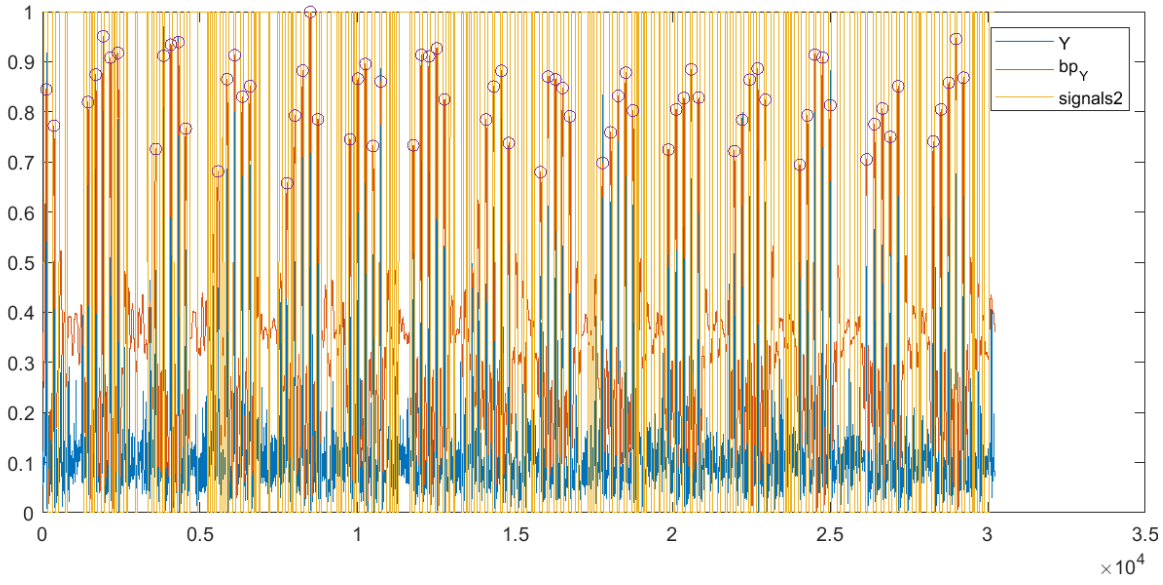
Algorithm 1 – tester 2:



Algorithm 2 – tester 1:

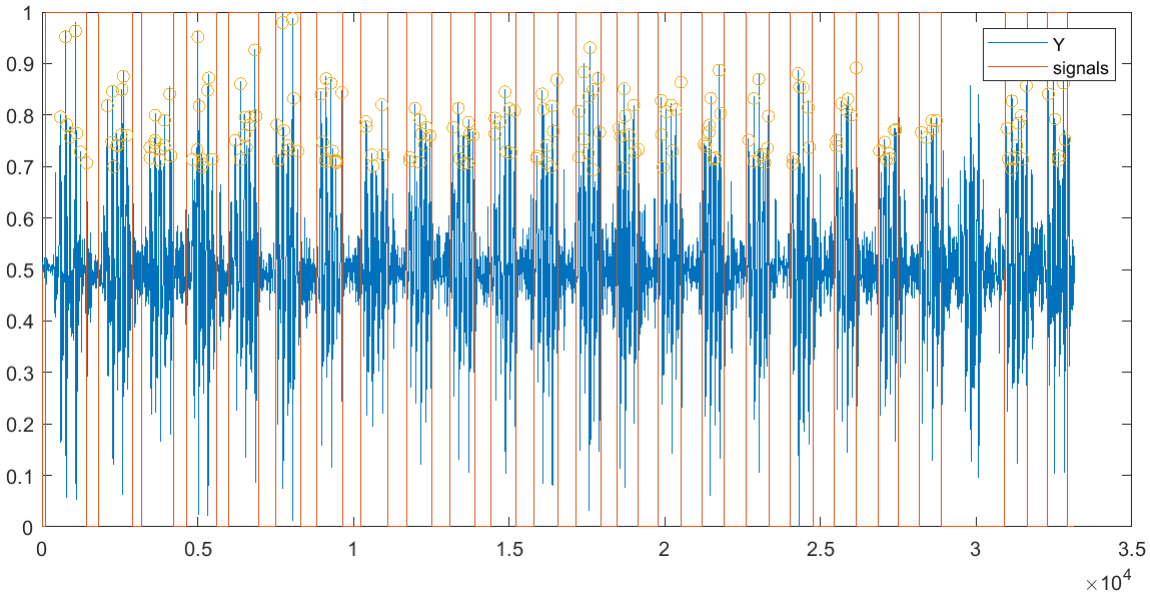


Algorithm 2 – tester 2:

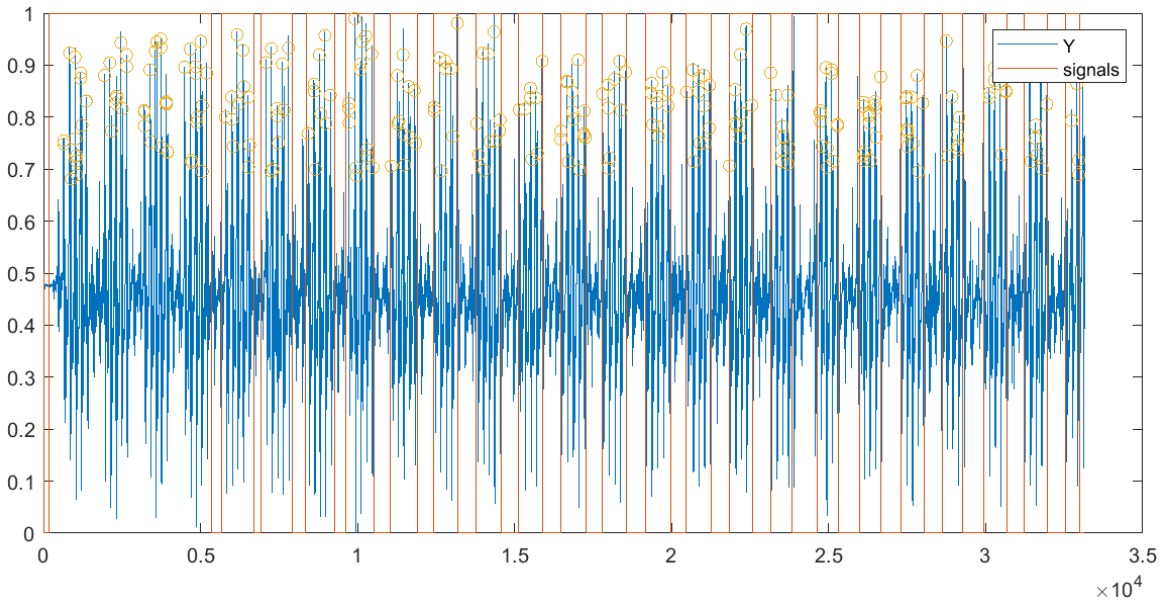


TEST 3

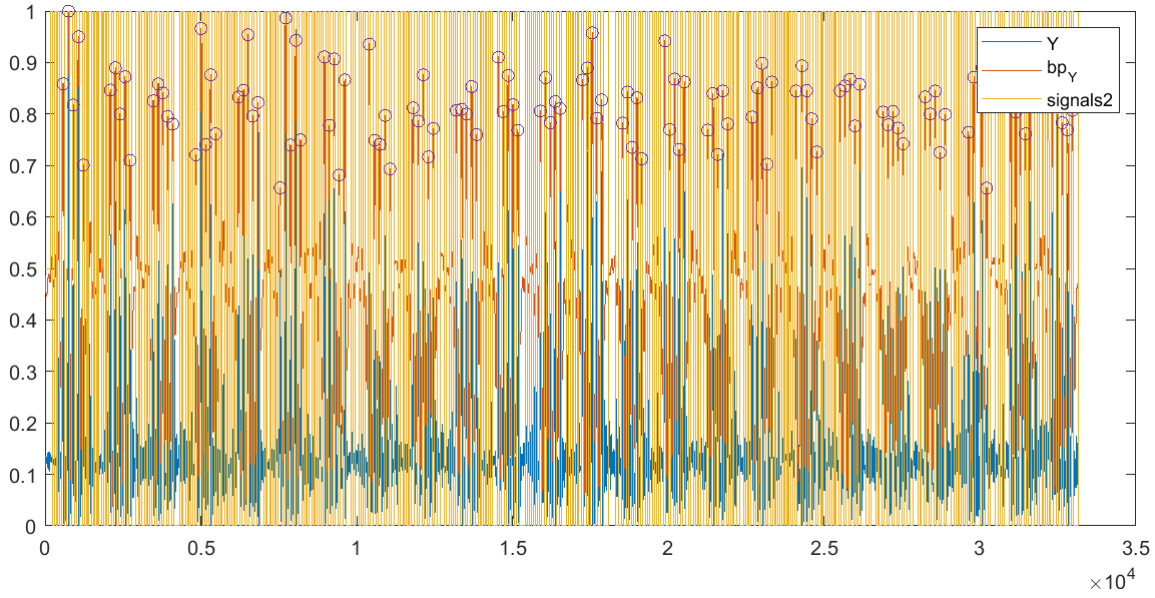
Algorithm 1 – tester 1:



Algorithm 1 – tester 2:



Algorithm 2 – tester 1:



Algorithm 2 – tester 2: