UBHI Prototype

Ciardi Roberto, Degiovanni Alessandro, Falzone Giovanni

Embedded Computing Systems, Sant'Anna School of Advanced Studies, Pisa, Italy

Contents

1 Introduction **3**
- 1.1 Purpose . . . 3
- 1.2 Prototype Overview . . . 3

2 Algorithm **4**
- 2.1 Introduction . . . 4
- 2.2 Description of the algorithm . . . 4
- 2.3 Filtering . . . 6
- 2.4 Tests, demo and conclusions . . . 7

3 Implementation **9**
- 3.1 Prototype realization . . . 9
  - 3.1.1 MPU6050 . . . 10
  - 3.1.2 Atmega328p . . . 11
  - 3.1.3 ESP8266 . . . 13
- 3.2 Algorithm implementation . . . 14
- 3.3 Real Time Analysis . . . 15

4 Middle Node **17**
- 4.1 Wearable module . . . 17
- 4.2 Web Server module . . . 17
- 4.3 Tasks organization . . . 18

5 WebApp **20**
- 5.1 Introduction . . . 20
- 5.2 Architecture . . . 21
  - 5.2.1 Communications . . . 21
- 5.3 Front End . . . 22
  - 5.3.1 Fall Detector . . . 24
  - 5.3.2 Other functionalities . . . 25
- 5.4 Back End . . . 27
- 5.5 Database . . . 27

1 Introduction

1.1 Purpose

This document describes the choices made to implement our prototype, that has the purpose of improving smart homes' healthcare. Smart home technology aims to support people to have a better quality of life and to ensure them to live comfortably and independently. As our main system, UBHI, represents a middle node between smart home and wearable devices, we decided to develop a prototype of a **fall detector**, that could be integrated in any wearable device and used in a smart home environment for people healthcare.

1.2 Prototype Overview

Falls in old people can lead to injuries and serious complications, so it is important to have a device that allows to detect these falls and notify them to helpers. Nearly half of these people cannot get up from falls on their own, and lying on the floor for an extended period of time can lead to serious problems: our prototype has the purpose to detect a person fall and notify it to the "helpers" contacts, as soon as possible.

Our prototype must be worn on the wrist, as a watch, with no difference between right or left, and can detect the motion and the position of the user thanks to two MEMS devices, a three-axis accelerometer and a three-axis gyroscope.

The prototype can be seen as the integration of three main parts:

- the **Wearable Device**, that with its sensors can detect rotations and accelerations and calculate falls conditions, and that communicates with the middle node via a WiFi connection.
- the **Middle Node**, that can communicate on one hand with the wearable device and, on the other hand, with the server of the WebApp via TCP connection.
- the **WebApp**, that gives a simple interface for the user and is useful to manage notifications.

2 Algorithm

2.1 Introduction

Considering the accelerometer and gyroscope data, the purpose of the algorithm is to detect a fall, studying the given signals. We firstly worked on MATLAB, in order to have visual descriptions of the signals and to study them in the better way; than, after a series of optimizations and tests on different data, we realized the algorithm in C code, to be processed by the Wearable Device.

To develop the algorithm to detect a user's fall, we made two assumptions:

- the prototype is worn on the wrist (right or left makes no difference), like a watch.
- the x-axis of the accelerometer and of the gyroscope are lined up with the forearm.

Moreover the algorithm shall have these two fundamental properties:

- accuracy: the algorithm shall be accurate for use in the home (i.e. it shall guarantee a 90% precision about sensitivity and specificity)
- simplicity: since the algorithm is executed in background on a low-power battery device, it shall be very simple.

2.2 Description of the algorithm

The algorithm we implemented is a two stages algorithm, based on the assumption made for some similar algorithms. [1, 2].

In the first stage the algorithm verifies the accelerometer and gyroscope values to check if there is a peak (i.e. an acceleration and an angular velocity much bigger than the usually detected values). In particular, a peak is detected if the following condition is true:

$$(A > 16 \;\&\&\; G > 1050) \tag{1}$$

with

$$A = \sqrt{A_x^2 + A_y^2 + A_z^2} \tag{2a}$$

$$G = \sqrt{G_x^2 + G_y^2 + G_z^2} \tag{2b}$$

where Ax, Ay and Az are the three components of the acceleration, and Gx, Gy and Gz are the three components of the angular velocity; the values of the accelerations are expressed in g (the acceleration of gravity), while the values of the angular velocities are expressed in deg/s.

If a peak has been detected, the algorithm processes the second stage; the objective of this stage is to verify if the values of the acceleration and of the angular velocity are consistent with that of an old person just fallen. In particular, we expect that, if the fall has been serious, the person lies slumped over the floor for a while. We verify this situation checking if the forearm is approximately horizontal as in figure 1, and without particular movements.

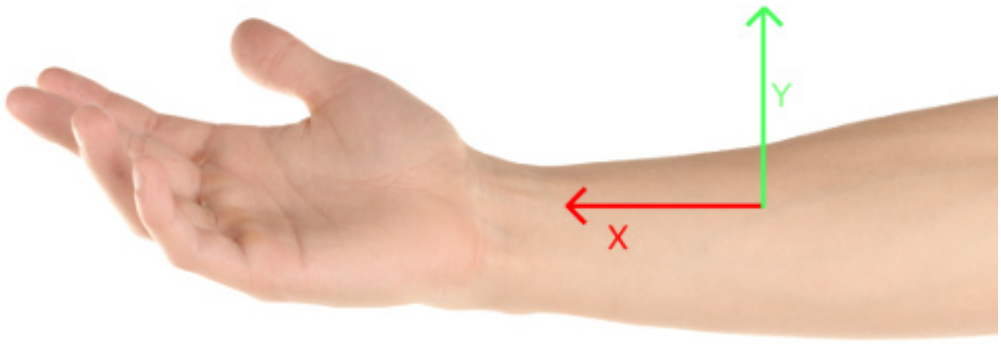


Figure 1: Hand position of a fallen man

More in detail, we check these conditions:

$$(A > 0.8 \;\&\&\; A < 1.2) \tag{3a}$$

$$(|A_x| < 0.4) \tag{3b}$$

$$(|G_y| < 90 \;\&\&\; |G_z| < 90 \;\&\&\; |G_x| < 180) \tag{3c}$$

where the 3a requires that there are not intense movements, the 3b requires that the forearm is approximately horizontal (the x-axis of the accelerometer is lined up with the forearm), and the 3c limits the values of the angular velocities. Experimentally we checked that the value of the angular velocity around the x-axis tends to be higher for a person that lies over the floor, so the last threshold is higher than the previous two. If the second stage condition is true, after a peak, for at least 150 samples among the 300 immediately after a peak (with a sampling rate of 50Hz), a fall is detected, otherwise the peak is considered a false alarm and the algorithm comes back to the beginning.

If a peak occurs during the second stage of the algorithm, the second stage of the algorithm restarts from the beginning (the last peak overwrites the previous peak).

So, at the end of the day, the algorithm notes a fall if there is a peak followed by a period of "calm", that represents the period in which the person is lying on the ground.

2.3 Filtering

We decided to not filter the signals acquired by the accelerometer and the gyroscope. This choice has been taken for the reasons that we are going to explain. First of all, we checked the state of the art of fall detection algorithms, described on scientific articles, and most of them describe algorithms that does not make filtering as we read in the documents mentioned before.[1, 2] On the other hand we saw that some algorithms use Kalman filtering.[3] However, the results we obtained with the previous technique were already good, and the computational complexity of the implementation of this kind of filtering is much higher than the computational complexity of our algorithm. In fact, implementing a Kalman filter requires to do matrix computations, while our algorithm is based only on the verification of a set of thresholds, that is much more efficient, above all for the execution on our processing unit.

We also tried to filter the signals with a FIR filter. This kind of filter (e.g. the moving average filter) is simpler than a Kalman filter and could be executed in our prototype without a huge computational effort. For example, a filter of this type is employed in applications like detection of user activities, as pedometer, that must recognize uniform patterns on a signal.[4] In our case the filtered signal was not a good approximation of the original one, because the peaks were all rounded, and thus it was more difficult to find the single peaks, useful to detect a fall.

In the following figure (2) are shown the norm of the acceleration for a fall, and the filtered acceleration with a moving average filter with a window size equal to 5; it can be easily seen that the peak has been significantly reduced, and becomes more difficult to detect.

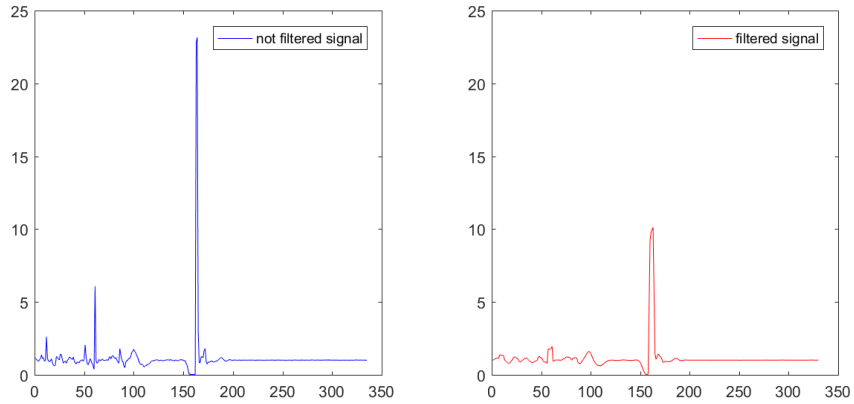


Figure 2: An example of moving average filtering

So, we decided to not filter the signal, to better detect the peaks, that are important for our analysis.

2.4 Tests, demo and conclusions

We made some tests with the hardware, saving the accelerometer and gyroscope's data on csv file and testing the correspondent signals on MATLAB and we saw that the algorithm had very good performances: we detected the falls we tested wearing the prototype on ourselves and also trying it on some deadweight (as a bag); however we saw that with intense impacts, followed by a stable positioning, parallel to the floor, could be recognized as a fall, having some false positives as for example when a user hit a table with his punch. This is mainly due to the fact that the prototype is put on the forearm and detect all the movements of the arm that, obviously, in normal active day, can be numerous. In ad hoc systems, in fact, a fall detector is usually put on the chest, to filter possible arm movements and to detect peak only on actual falls. Anyway, these outliers are unlikely (during our tests more than 95% of the impacts were interpreted correctly), and if the device detects a fall, it signals it to the user with a blinking led, so that the user can confirm to be all right pressing a button. Furthermore, if we consider that the prototype should be worn by an old person, it is very unlikely that he make rough movement, unless he actually falls.

In the demo we made, we tested the prototype on a doll; since the elbows are rigid, when it is lying down, its arms are not in a natural position for an old person just fallen, so we put the prototype on the leg (in particular on the thigh): in this way, when the doll has fallen, the x-axis of the prototype

is approximately parallel to the floor, as it would happen for the forearm of a real person just fallen. When we tested the demo, moreover, we noticed that the thresholds we had chosen, testing the prototype on us, worked well also with the doll: in fact, even if the doll is significantly lighter than a person, it is quite rigid, so it does not reduce the impacts.

3 Implementation

3.1 Prototype realization

Different versions of the prototype have been realized in order to record some useful signals, needed to test and tune the algorithm; the last version of the prototype is wearable, to simulate the actual utilization of the device: the hardware have been fixed on a pcb by soldering strips and wires, also two buttons and one led have been inserted in the prototype to signal some special events like a fall or to send a predefined request to the middle node.

In the figure 3 a picture of the final version of the prototype is reported. The components of the prototype are:

- MEMS MPU6050 Tri axis Accelerometer and Gyroscope
- MCU Atmega328p
- WiFi ESP8266

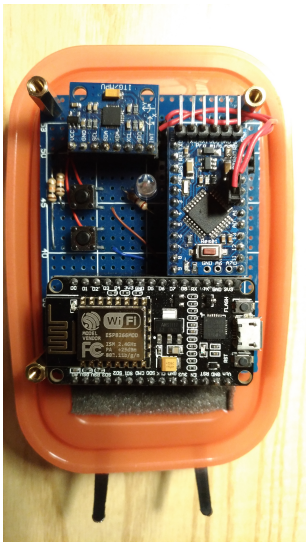


Figure 3: Wearable prototype

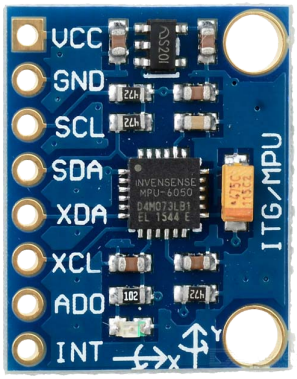


Figure 4: MPU6050 breakout board

3.1.1 MPU6050

The InvenSense MPU-6050 sensor contains a MEMS accelerometer and a MEMS gyroscope in a single chip. It is very accurate, as it contains 16-bits analog to digital conversion hardware for each channel. Therefor it captures the x, y, and z channel at the same time. The sensor uses the I^2C-bus to comunicate with an application MCU.

For precise tracking of both fast and slow motions, the parts feature a user-programmable gyro full-scale range of ±250, ±500, ±1000, and ±2000 °/sec (dps), and a user-programmable accelerometer full-scale range of ±2g, ±4g, ±8g, and ±16g.

For our purpose the full-scale range used for both accelerometer and gyro takes the maximum value ±16g and ±2000°/sec; the MPU is powered by the A328P with 3.3V and connected via I^2C running at 400kHz.

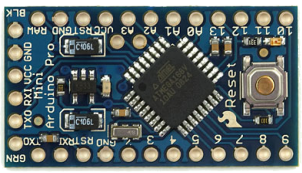


Figure 5: Arduino Pro mini

3.1.2 Atmega328p

The Atmega 328p is a 8-bit AVR RISC-based microcontroller with 32x8bit registers, it combines 32KB flash memory with read-while-write capabilities, 2KB SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three timers with compare modes, internal and external interrupts, serial programmable UART, a byte-oriented 2-wire serial interface, SPI serial port, a 6-channel 10-bit A/D converter. The device operates between 1.8 - 5.5 volts.

The board used is an Arduino pro mini that uses the Atmega328p microcontroller running at 8MHz and with a working Voltage of 3.3V. This controller comunicates with the MPU6050 by means of the I^2C running at 400KHz; the data read by the MPU6050 is elaborated by the algorithm implemented in C, then those data are sended to the WiFi controller by UART running with a baudrate of 115200bps. Both the speed of I^2C and the baudrate of UART have been taken at the maximum value in order to minimize the time required by the comunication for each execution of the algorithm.

The data sent to the middle node is composed in this way: "S»-123456,-123456,-123456,-123456,-123456,-123456//" where the "»" is a delimiter to separate the values by the command and the "//" is used to separate two different lines, whereas the values are organized in this way: "A_x, A_y, A_z, G_x, G_y, G_z". All the numbers are converted in their string representation on 6 chars, so the exchanged bytes for each sample is of 52 bytes.

Different messages are sended by the microcontroller according to other events as reported in the following list:

- "S»//" is used to send a Sample of motion
- "R»//" is used to send a request to Record the buffered samples in the middle node
- "W»//" is used to send the name and id of the user and of the device
- "A»" is used to send a signal to notify that the user is "Alive" and the last fallen signal can be ignored

Starting from the code realized and tested in MATLAB, we have coded in C the same algorithm; a functional block description of the workflow executed

by the A328P is reported in the figure 6. The code is realized in order to execute the algorithm at 50Hz; for this purpose a pseudo RealTime task is realized by a branch of the main code that run in loop at higher frequency.

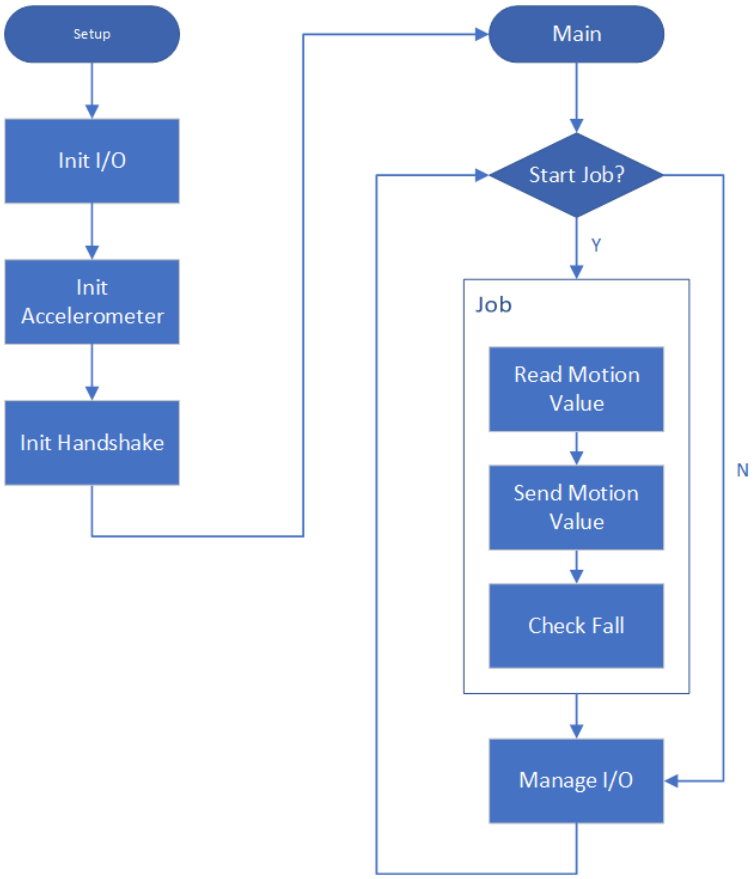


Figure 6: Workflow of the A328P

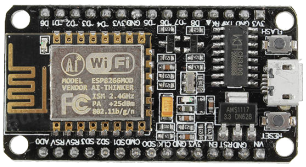


Figure 7: Nodemcu esp8266

3.1.3 ESP8266

The ESP8266 WiFi Module is a self contained SOC with integrated TCP/IP protocol stack that can give access to WiFi network. The role of this controller is to realize and manage the reliable communication with the middle node.

In the figure 8 is represented the workflow realized by this component, in the setup phase the connection to the WiFi network is enstablished and then a TCP connection is realized by means of a TCP socket that establishes the connection with the middle node, the esp8266 receives messages by the A328P via the UART interface and send it through the TCP socket.

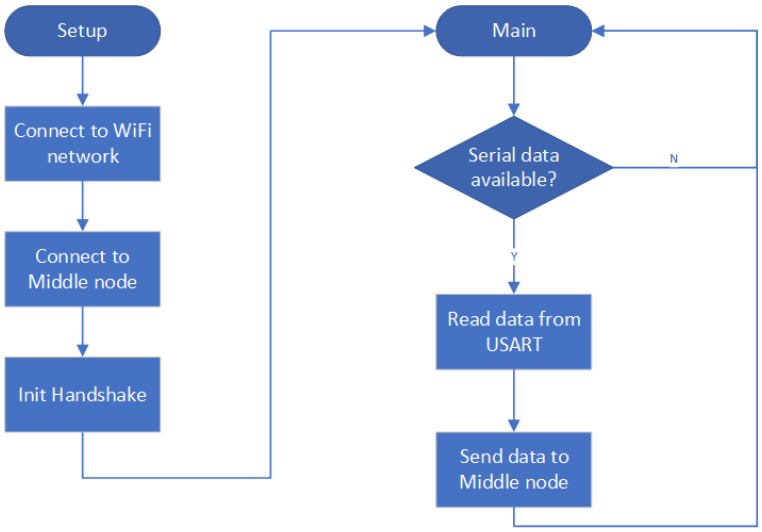


Figure 8: Workflow of the ESP8266

3.2 Algorithm implementation

The C code implementation of the algorithm have been realized reducing the computational power required by the algorithm at the minmum level, in order to do that all the threshold have been converted according to the used full-scale range by the preprocessor; another important consideration is the fact that thanks to this trick all operation are between integer and not real numbers. Also the square root required by the algorithm have been eliminated considering the square of threshold computed by the preprocessor.

In the following is reported a pseudo code to describe the algorithm executed by the MCU.

```
checkFall(motion) :
  if(check_Acceleration_Peak && check_Gyro_Peak) :
    Peak_Detectd
    Start Timeout

  if(PeakDetected) :
    if (!Timeout_Fired) :
      if(Check_Dying_Acceleration && checkDying_Position) :
        Count_Dying++
      if(Count_Dying > DYING_THRESHOLD) :
        Fall_Detected
    else :          // False Alarm
      Reset_Timeout
      Reset_Peak_Detected
```

3.3 Real Time Analysis

As said the algorithm is running at 50Hz. To verify the feasibility, the system has been studied for 10 000 iteration and the data regarding the starting time and the execution time of each job have been collected and analyzed to check the feasibility.

In the figure 9 is reported the distribution of the difference between the activation time and the start time, whereas in the figure 10 is reported the distribution of the execution time; the values are reported in μs to have a better resolution; through these plots is easy to see that the timing constraints are always followed.

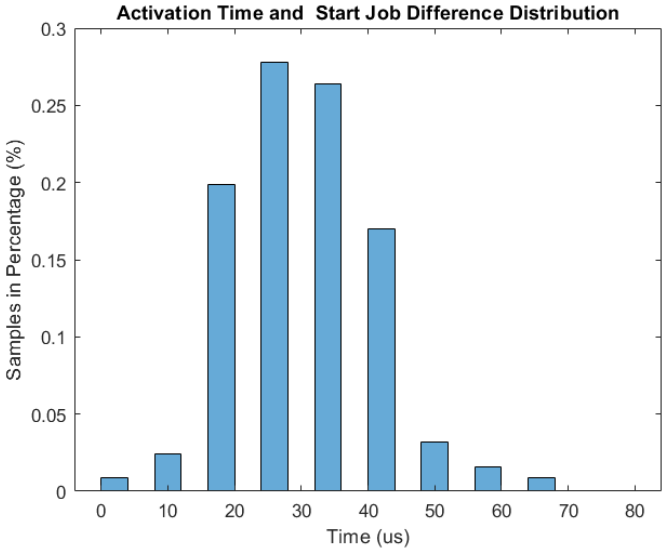


Figure 9: Activation Time and Start Job difference distribution for 10 000 cases

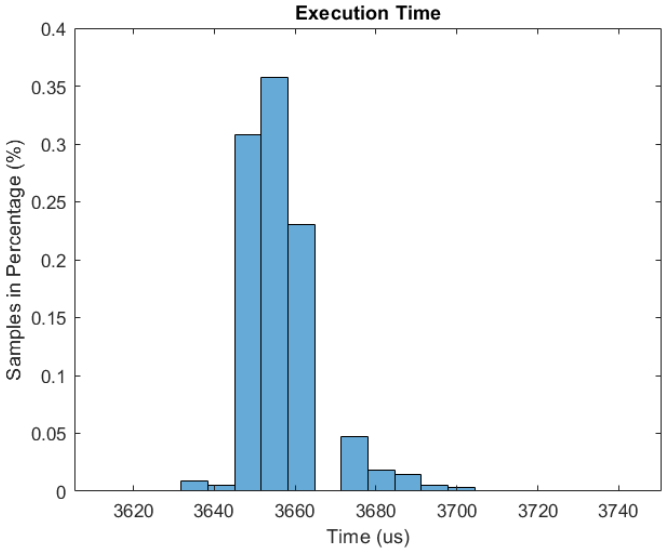


Figure 10: Execution Time distribution for 10 000 cases

4 Middle Node

The middle node is composed by two logically separable modules:

- the direct connection with the wearable device
- the direct connection with the web server

4.1 Wearable module

In order to realize the connection with a wearable device, an entity, that we can call "Wearable Manager", acts as a server, listening on a binded socket, to serve all the possible wearable devices that would like to connect with the middle node. It is thought as the middle node of the UBHI system, that could interact with more wearable devices, even if in our case it has to interact only with our prototype. Once the wearable device has initiated a connection reqeust, the Wearable Manager creates a new entity that takes the control of the socket -we can call this entity "Wearable Fall Detector"- this entity is obviously device dependent, in fact it realizes the functionalities of the device. Once a Wearable Fall Detector is created, the associated socket is used to receive the messages sent by the wearable device -in our case the received strings are exactly those sent by the A328P- and finally, the string is parsed to obtain the command and data. For each command the correspondent operation is executed, while, for the sample one, a circular buffer of the last 60s is mantained, to eventually store it. When a fall is detected by the device, a fall command is sent and the Wearable Fall Detector creates another entity that, after a timeout of five seconds -useful to cancel the fall with an "alive" command-, sends a message to the Web Server to notify the event; if an Alive command is sent before the firing of the timeout the procedure is stopped.

4.2 Web Server module

The connection with the Web Server is realized by a periodic task, that we can call ”Web Server Manager”. The role of this task is to empty a buffer containing all the data sent by the wearable devices: when data are available, the task gets them - that could belong to different devices for an expansion of our prototype- and send them as a JSON string to the Web Server, through a HTTP POST request. A different behaviour is set when a file containing the last buffered samples has to be sent: in this case the task sends the file as payload of a HTTP POST request to the server.

The middle node has been relized in Python, and each entity is a class instance that implements a Thread; most of them are periodic tasks, but the

blocking calls of course stop the execution of the task when no action can be performed. The timeout event on the socket, instead, is used to guarantee that a not responding device does not occupy resources of the system and, to reconnect, it have to establish another connection with the middle node.

4.3 Tasks organization

In the figure 11 is reported the tasks organization of the system; the circles represent threads, while the square represent the class that realize the buffer used to send messages by the devices and to get data by the Web Server Manager, in the figure 12.

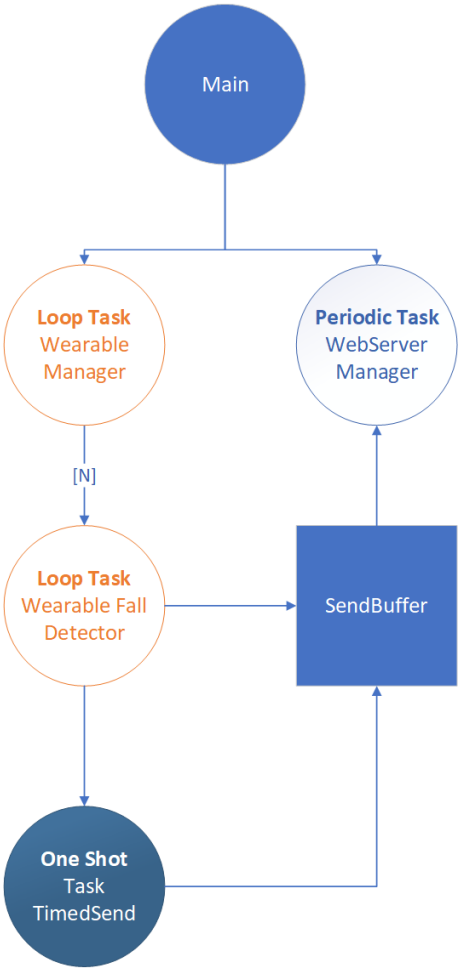


Figure 11: Tasks organization for the Middle Node

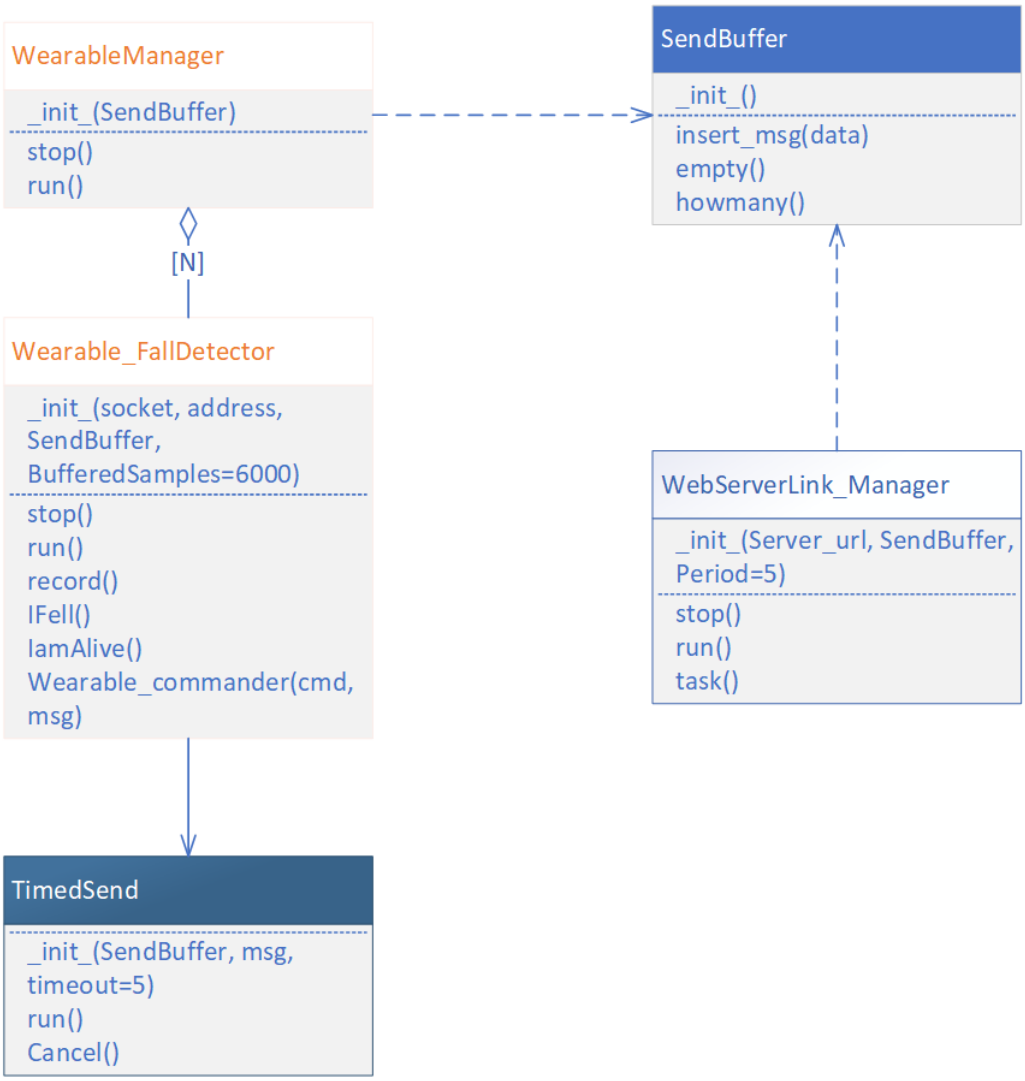


Figure 12: UML description of the Middle Node

5 WebApp

5.1 Introduction

For an informative system it is very important to show in a clear way the collected data and make them accessible to every type of user, giving simple feedback through the most familiar devices. In particular, for our prototype, we needed a real time system to simulate the notification system, useful for first aid, needed when a fall is detected. For this reason we developed a web app that presents our prototype with a nice and responsive way, giving real-time feedback and allowing to study prototype's data.

Our web application follows the classical approach, it is a client–server computer application which runs mostly on all browsers, on PC and also on smartphones. The client includes the user interface and client-side logic while the server-side is in charged to retrieve and organize stored data to present it to the client-side logic.

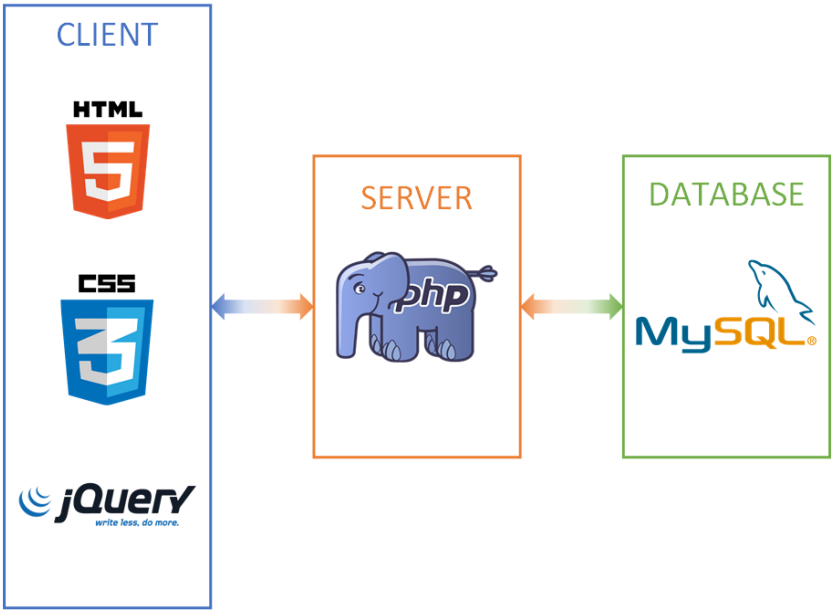


Figure 13: Webapp architecture

5.2 Architecture

The architecture of the web application is based on the classical architectural pattern of web application. It is hosted on altervista.org - visitable at - a free-hosting that permit to get a DB and a PHP engine for free.

As we can see in the figure 13, for the Client part we used the well-known markup language HTML5, with CSS3 and jQuery, a very useful cross-platform JavaScript library; for the Server part, on the other hand, we used PHP5 (supported by Altervista) with a Database MySQL 5.6.

5.2.1 Communications

The communication between PHP engine and the database is realized by means of mysqli, a standard component of PHP that implement some functions to perform queries on database. The communication between client end -running on browser- and PHP scripts -running on hosting server- is build through HTTP, mainly with POST requests. Some data between client and server are sent in JSON format, widely used nowadays as a replacement of XML.

5.3 Front End

The client side is organized in two main parts: the homepage, in figure 14, that is a sort of presentation page, and the app, that is the core of the application.

Figure 14: Responsive Homepage

The homepage is divided in three main parts: our team, where we present ourselves; our idea, where we present the main UBHI idea, with its characteristics; and our prototype, where we say a few words about the fall detector we implemented.

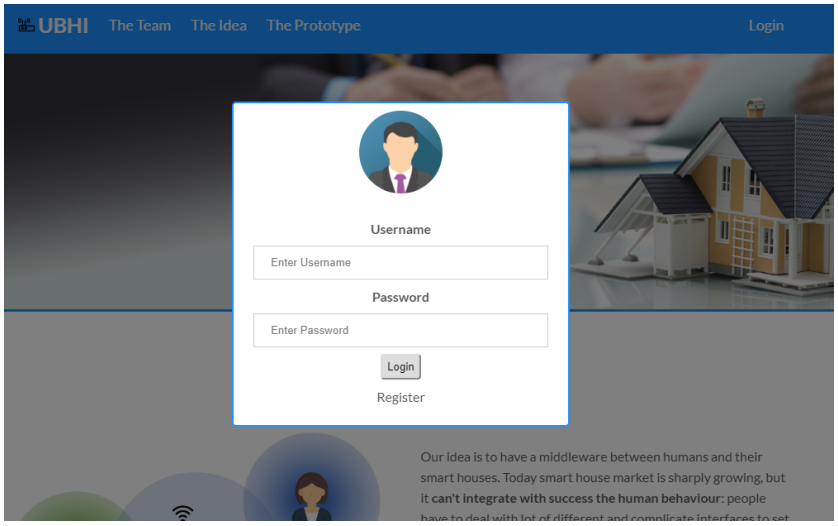


Figure 15: Form to Login

On the top right of the page you can click on "Login" button, as in figure 15, to access to the app where you can manage your devices: the app is thought to be the UBHI app, so it should be used by an user who owns a wearable device and some smart home devices and who can have an overview of these device and manage them with its smartphone. Obviously in this case we implemented only the Fall Detector part.

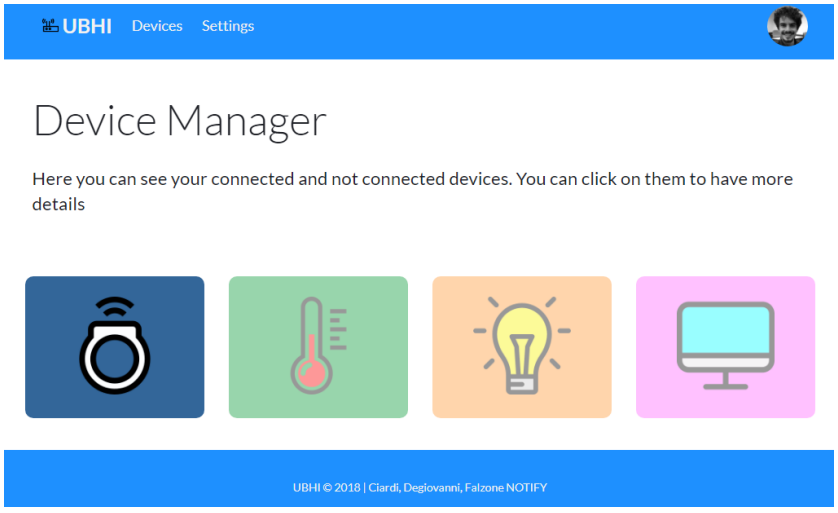


Figure 16: Main page as logged user

As you are logged you can see your devices, as in figure 16, and manage them; as said, in this case, only the fall detector is enabled and clickable.

5.3.1 Fall Detector

The fall detector page, as in figure 17, is useful to display to user the signals sampled by the device.

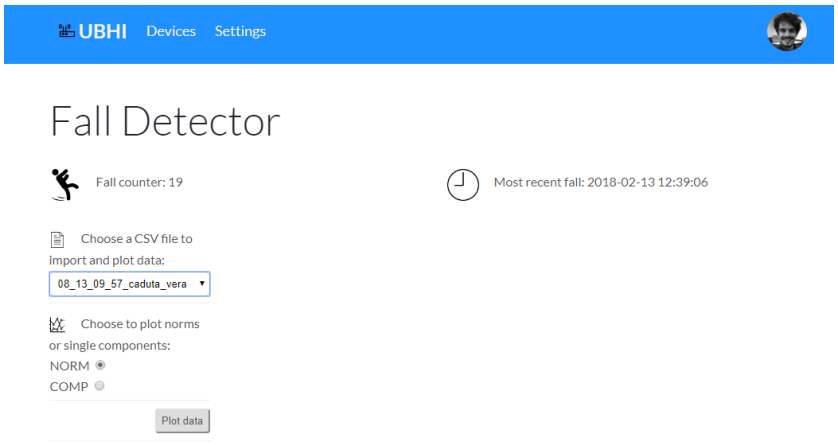
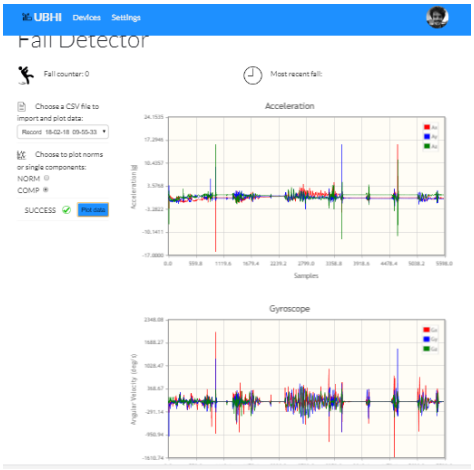


Figure 17: Fall Detector Page

Each user here can see how many times he has fallen and when it was his last fall. For our main tests we created an account for our baby, Gionni, who falls more times than us. As explained before, from the fall detector, the signals can be saved on csv files that can be plotted in this page: on the left side there is a selector where you can choose a file - the files that contains a fall are called "Fallen_date_time.csv", whereas the others are "Random_date_time.csv" - then, with a radio button, you can choose to plot the norm of acceleration and gyroscope or their single components. Clicking on "Plot data" button the signals are plotted on the right side of the page, as we can see in the figure 18.

(a) Plot of accelerometer and gyroscope norms



(b) Plot of single components

Figure 18: Here we can see the two type of plots

To draw the plots we used **jqPlot**, a jQuery plugin that allow to draw and redraw plots on canvas, allowing us to have the results shown in figure 18. Obviously a zoom functionality allow user to have a clearer view on the plot that should appear as very complex.

5.3.2 Other functionalities

As a logged user there are some other accessible functionalities: in particular, from settings page, you can specify which are your "favourite contacts", the people that must be called if you fall.

Very important is also the notification system: when a user is logged, if one other user, that specified the contact of the first, falls, a notification as the one in the figure 19 is shown, blinking for thirty seconds. The service also send an email to all the contacts and a natural expansion of the system will be a notification via SMS, eventually with a call to ER.

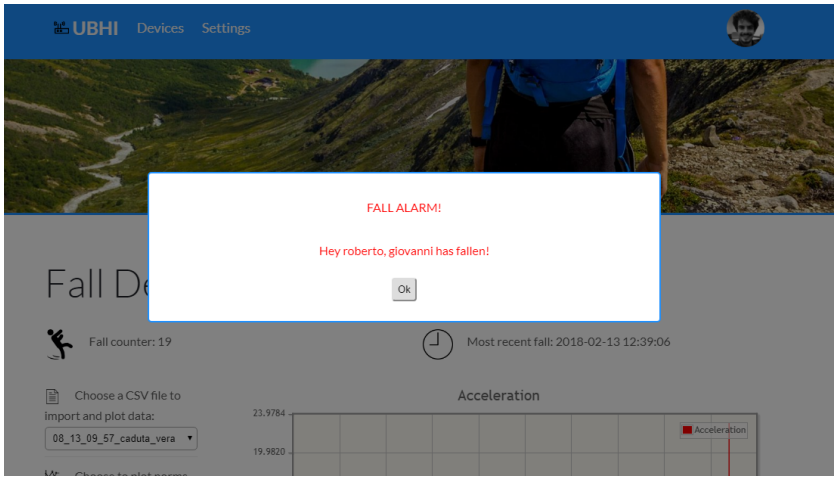


Figure 19: Notification Banner

5.4 Back End

The main functionalities of the back end part of the app were already explained talking about the frontend: obviously it must interact with database, allowing user to create an account and login in the app, but also specify its contacts. It is very important the interaction of the server with the fall detector device, that, as explained in section 4, can send data with POST request, as JSON or csv files. If a JSON arrives the web server control its content to understand if it represents a fall, in such case the server register the fall in the database - with the fallen user that is specified in the received message - and performs a query to the database to find and alert the help contacts of the fallen user. In the case that the server receives a csv file from the device it register it on the database and saves the file on the server, to be plottable in fall detector section of the app.

5.5 Database

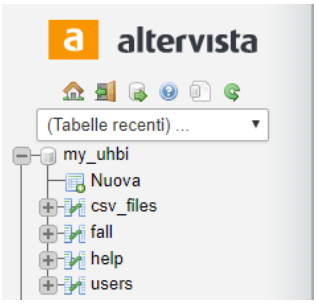


Figure 20: Database organization

As already underlined, the database was implemented with MySql 5.6 and hosted on altervista. It has a very simple structure with four tables, as seen in figure 20: "users" that contains the username and the password of the registered users, that can be added from the app; "help", that contains the help contacts of each user, if specified, and also in this case they can be added from the app; "fall", that is used to register the falls events; and "csv_files", where the name of the uploaded files, containing the signals, are maintained. An extension of the app should see the presence of a "device" table, to associate multiple devices to each user.

In the figure below 21 a simple schema of the database is given.

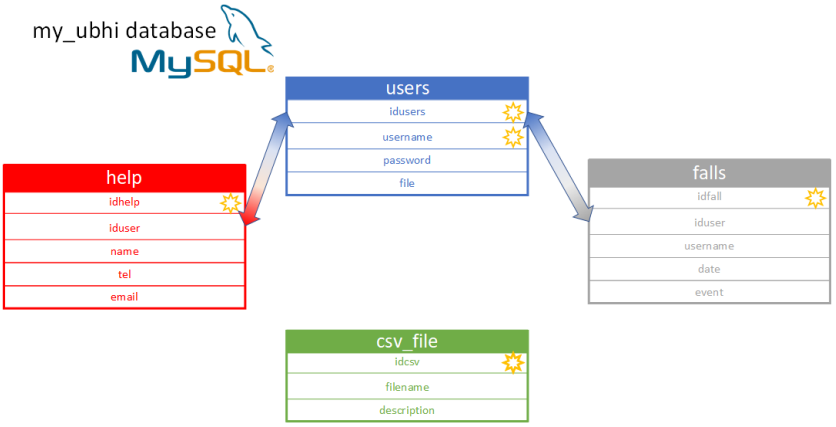


Figure 21: Database Architecture

A more detailed analysis can be made on the single tables:

- **users** contains, as key, the id of the user and his username, whereas the password can be anyone chosen and the file contains the name of the profile image, if uploaded while registering.
- **help** is the table containing help contacts, each entry is associated to a user - it is one of is contact - and contains the username, the email and the phone number of the contact.
- **fall** is the table where the falls are saved, each of them associated to a user by its id and username. This table can contain also other signal coming from the device, so an event column is important to know if that signal was a fall.
- **csv_files** contains the name of the file uploaded on the server an plottable on the webapp.

References

[1] *A Multistage Collaborative Filtering Method for Fall Detection*, Tao Xiea,b,c, Yiqiang Chena,b,c, Lisha Hua,b,c, Chenlong Gaoa,b,c Chunyu Hua,b,c, Jianfei Shena,b,c a Beijing Key Laboratory of Mobile Computing and Pervasive Device, Beijing, China b Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China c University of Chinese Academy of Sciences, Beijing, China.

[2] *Detecting Human Falls with a 3-Axis Digital Accelerometer* Ning Jia.

[3] *An Unobtrusive Fall Detection and Alerting System Based on Kalman Filter and Bayes Network Classifier.* He J, Bai S, Wang X.

[4] *Detecting User Activities using the Accelerometer on Android Smartphones*

[5] *UBHI webApp*