UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia “Galileo Galilei”

Master Degree in Physics of Data

Final Thesis

Study of a wearable IoT solution for personal safety

Internal advisor

Prof.Alberto Garfagnini

Internship supervisor

Dott. Francesco Vinelli

Candidate

Tommaso Tabarelli

Academic Year 2019/2020

Abstract

Recent advances to internet infrastructure have allowed an unprecedented expansion in the number of devices connected to the network. Smartphones, tablets and computers are only some of the devices that participate to the global network. Internet of Things (IoT) is a neologism which refers to the extension of the Internet to common objects. A great deal of interesting applications have flourished that take advantage of the possibilities offered by this expansion.

In this report, we document the process of studying and developing a wearable, integrated solution which uses a wide range of sensors in order to monitor some environment parameters. The scope of this device shall be detecting and alerting the user about nearby possible dangers; some use case examples will be showed and some possible real simulation will be explained. Sensors can be applied to different wearable elements (i.e., a vest, a smartwatch, gloves, a helmet). The gathered information is transferred to a back-end system where a specialized analysis is carried out and potential actions to safeguard user health are taken. Data and notifications are meant to be sent to a control room where monitoring people are always ready to take appropriate actions. Some test data are analyzed to give an idea of the potential information data can have; other series are generated to test the algorithm functioning.

Contents

Chapter 1

Introduction

This project is developed by the *Emerging technology* team of *PriceWaterhouse&Coopers* for the *European Immersive Computing Summit* (EICS) [2] and for *Smart Production Solutions* (SPS) [3] exhibitions.

1.1 The firm: PwC

PriceWaterhouse & Coopers is among the world's leading professional services networks. Relying on offices located in 157 countries all around the world and counting more than 276000 employees, PwC is a global network providing services in different professional areas: assurance, tax, advisory [1]. Founded in 1998 by the merger of *Price Waterhouse* and *Coopers & Lybrand*, PwC vaunts more than one hundred and fifty years in professional services, since **Samuel Lowell Price** set up his business in London in 1849 and **William Cooper** started his in 1854. Since then, their activities faced many collaborations and mergers, growing and expanding their fields of activity, finally merging together.

Emerging technology is a new team built to face new challenges the contemporary business environments and innovations are requiring. In this perspective, PwC is financing IoT projects to show that it can manage projects in this novel field, achieving good results. This purpose embrace the present work, which is to be presented at *EICS* and *SPS Italia* exhibitions to show that PwC is innovating itself and is ready to compete in the technology niche.

1.2 The exhibitions

European Immersive Computing Summit (EICS) is a European conference gathering entrepreneurial, technical and creative minds to build the future and spread the word about their work. Its aim is to gather the leading names in emerging technologies to participate to workshops, business meetings and to share inspiring keynotes. Usually, there are practical case studies from corporate, agencies and startups giving real examples on how to implement these new cutting-edge technologies into projects and business. It represents an opportunity to get an overview on tech solutions in the field of manufacturing, mobility & smart tourism, healthcare, architecture and design, art and entertainment, retail. [2]

Smart Production Solutions (SPS) is a Italian exhibition devoted to smart, digital and flexible industry. Since 2011, it is a key meeting to unearth and discuss the main themes involving the industry of the future. Among SPS 2020 main topics there are additive manufacturing, artificial intelligence, augmented reality (AR), automation, big data, blockchain, cyber security, digital twin, green manufacturing, robotics, smart machines, and many others [3].

These two opportunities are key test benches for the firm to verify the impact of its projects and the feasibility of its ambitions.

Chapter 2

The project

The assigned project involves the development of a multi-sensor IoT solution for personal safety. The main idea is to exploit a Raspberry module, a smartwatch, some smart glasses, which are small enough to be integrated in a wearable suite without causing much encumbrance to the wearer. Many sensors are applied to the wearable so that environment information, such as temperature and presence of gas leaks, can be detected and gathered. Data collected from different sensors is sent to remote server in distinct ways, depending on the sensors it comes from. At server level data can be stored and better elaborated than at local level.

The scope of the work is to show that in principle it is possible to build a compact wearable solution using different kind of sensors and items, including smartwatch, smart glasses; furthermore, there is the possibility to include new sensors in future. The resulting object is integrated with a back-end system and with the human intervention (e.g.: remote human assistance for in-loco first aid interventions). It should be able to detect risks, to send alerts and to help protecting people safety from the possible dangers they face during their work routine or in special situations, such as accidents involving people. The back-end system collects and elaborate the information, sending back notifications to user devices when they finds themselves in a dangerous state or sending notifications to nearer users in order to make them help people facing troubles.

The main points of the project development involve:

- *Use case* analysis and possible simulations
- sensor description and analysis
- data collection and data management
- data simulations and algorithm analysis
- application ideas and possible improvements

My tasks in this project were to verify the Raspberry module configuration, to check sensors functioning, to understand how they collect the data and how they interact with the Raspberry module, to write a program to collect data from different sources simultaneously (example scripts are given, but they use only one or two modules at a time; for this project, many of them are needed at the same time). Furthermore, to better realize my assignment, an understanding of the whole system was needed. Finally, some simulation were made and analyzed to test the back-end algorithm.

Chapter 3

Use Case analysis

In this chapter main *Use Cases* will be analyzed and many possible applications of the wearable solution will be discussed.

3.1 Assumptions for Use Case

Chosen field sensors may not give accurate parameters estimations (such as exact temperature or exact gas density). Rather, they are expected to measure the parameters basing on arbitrary but reasonable thresholds which can be used to detect dangers. Taking as an example the temperature parameter, the main goal is not to detect the exact temperature but rather to notice if the it is larger than about 35 °C (example value) to make the system able to notify if the user from which the measure came from is in a possible danger situation. In other words, the scope of this work is to qualitatively detect dangers, not to quantify them; in a sense, they are already quantified and considered risky when the measure threshold is exceeded. For this reason, sometimes the exact method of sensor data transformation into a number is not explained (some information lack also on the vendor guides and websites, so they must be looked for in external sources).

For similar reasons, data from the Smartwatch sensors (accelerometer, gyroscope, altitude, heart rate) is taken *as is*, since for some of them it is not possible to know the exact way they work.

Some users can be provided cutting edge technology items, including *smartphones*, *augmented reality viewers*, *smart glasses*. A **data integration platform** supports the whole system collecting information coming from the different sources (environment sensors, smartwatch sensors) and making them homogeneous.

Flux of information will be bidirectional between users and the platform: first step is data collection from sensors; after having elaborated them, if necessary, notification can be sent to user devices. Following IoT paradigm, signal administration logic will be implemented to collect data regardless the type of sensor they come from. Those records will be exploited to remotely manage people in the work area.

3.2 Shared features

Basically every user can be notified by a smartphone or via different kind of notifications depending on the equipment the person is bringing. If a user has a smartwatch, the notification can be shown also on it. Furthermore, if a user is wearing smart glasses, a overlay visual notice can be shown on them.

User can test various kinds of notifications related to different scenarios. A variety of distinct events can be simulated to test data collection from sensors and its elaboration. In particular, acquired data will be related to:

- Environmental information
    - gas leak monitoring

  - flames and fires detection
  - excessive noise detection

- Indoor and outdoor location
- life parameters real time monitoring

This kinds of Use Case are at the foundation of the interactions and the realization of an immersive experience, aiming to recover absent-minded workers. In the following, many Use Cases are proposed to show how effectively the presented system can lead to relevant and actual improvements during dangers handling, but also in space and people management.

3.3 Use Cases hypothesis

3.3.1 Dinamic Virtual Wall and Group Notifications

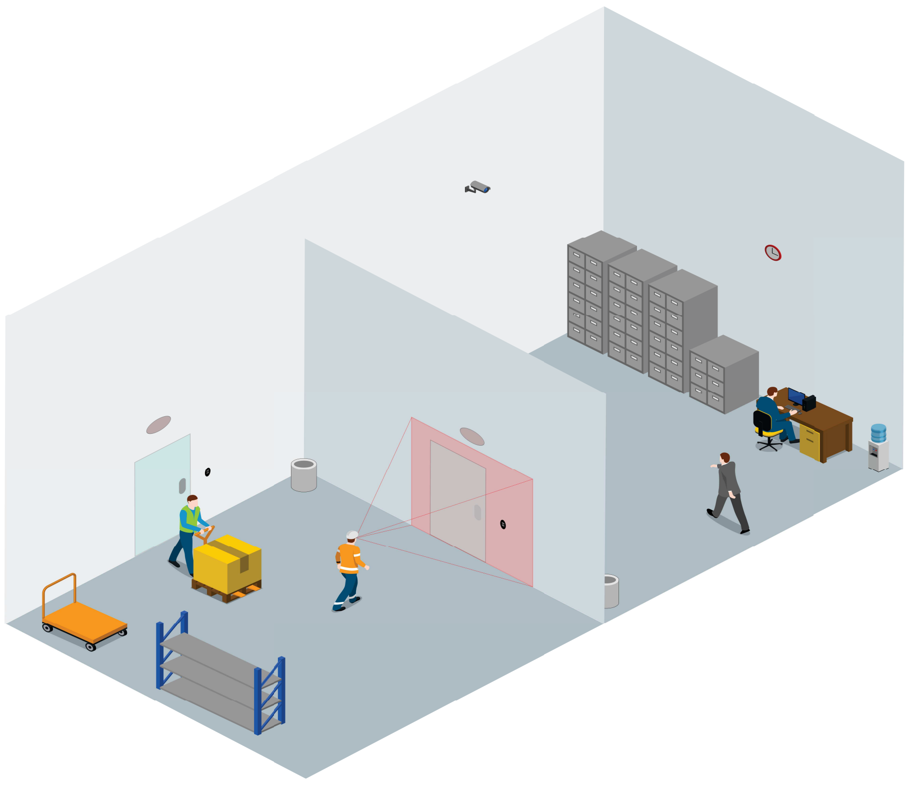
In this context many access management situations involving the admission to physical work spaces in alert circumstances and in standard conditions are explained. The goal is to show how, making use of latest IoT and AR software and solutions, it is possible to avoid personnel overlaps, to prevent non-authorized accesses, to handle complex situations as emergencies and accidents, having a global vision about every business process from a single control center. The users will be real-time instructed on how and where to move in work area basing on the actual circumstances. It is assumed personnel diversification based on roles and competence. Every person belonging to different groups will be instructed on what actions they may or may not do.

Simulation Users will be asked to move in a work space (previously divided in different sections, which are classified in different types). When someone is found going to a zone he might not approach, notifications to not proceed will be sent to that user to advise him he can not access that area. Notifications type can change also basing on the devices the user is wearing/using (e.g.: visible feedback on smart glasses). First kind of simulation is based on a virtual wall delimiting the space a user can access basing on the permission that person has (Figure 3.1). Then more complex simulations can be done, involving fires or flooding situations.

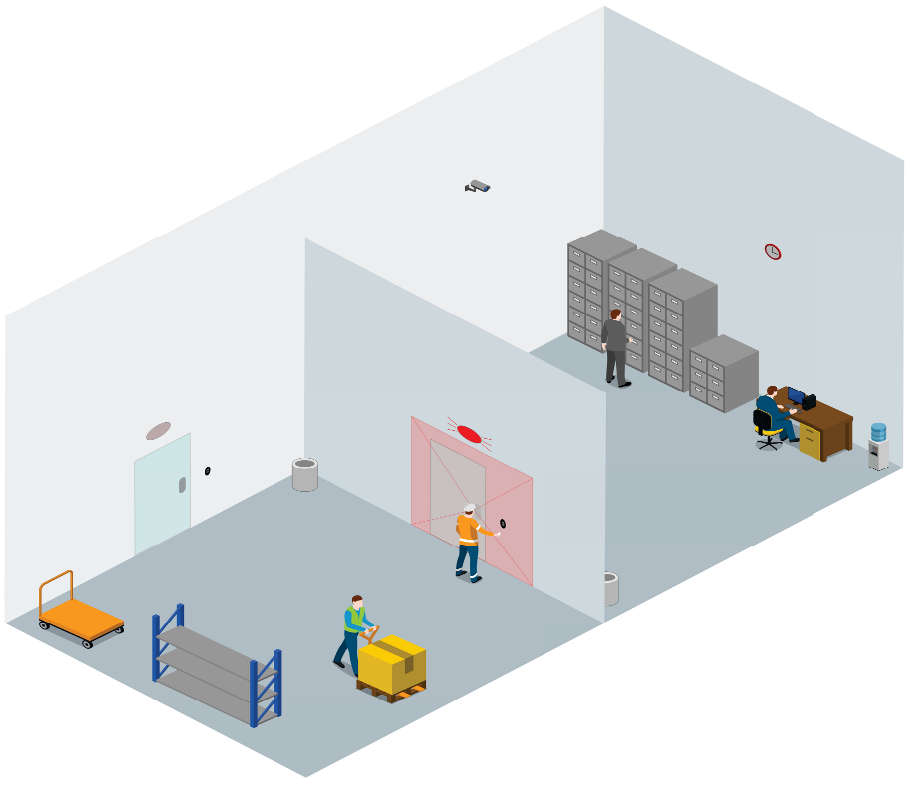
Example: fire During a fire event there are two main responses to act: the first is leading the users to safe zones and the second is to take care they arrive to safe zones without choosing the wrong path. To achieve this scopes, smartwatches will be used, eventually combining them with the other objects the users have (smartphone, smart glasses, tablet). From a real environment, a fire event will be simulated, enlightening how smart management of people in the environments can simplify gathering processes towards safe zones, thus lowering the dispersion of users on the way of chaotic environments. Users movements will be easier due to the items they are equipped with and they will not be asked to take decisions basing on their instinct, avoiding fire environments thanks to the feedbacks they receive. At the same time, those who are qualified to approach the novel fire will be notified about the nearest means they can use. In this case, walls will be used as beacon for the fire environment to make people reach it faster; these notifications can be combined with those on the other objects.

3.3.2 Classified Man-Down and First Aid

In this case the focus is put on personal safety and accidents management on workplace, called "man down" events; these have to be handled basing on the different circumstances which originate them. The goal is to show the possibility of timely intervention in case of injury, exploiting AR solutions, IoT paradigm, smart objects. It is assumed peronnel diversification basing on roles and competence, with particular focus on first aid abilities. Based on this information, on people position and relating on the *man down* user location everyone will be call to act in the proper moment, helped by remote assistance.



(a)

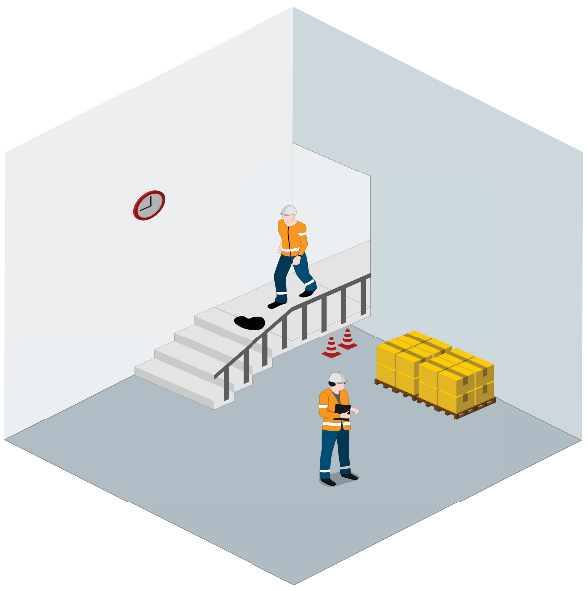


(b)

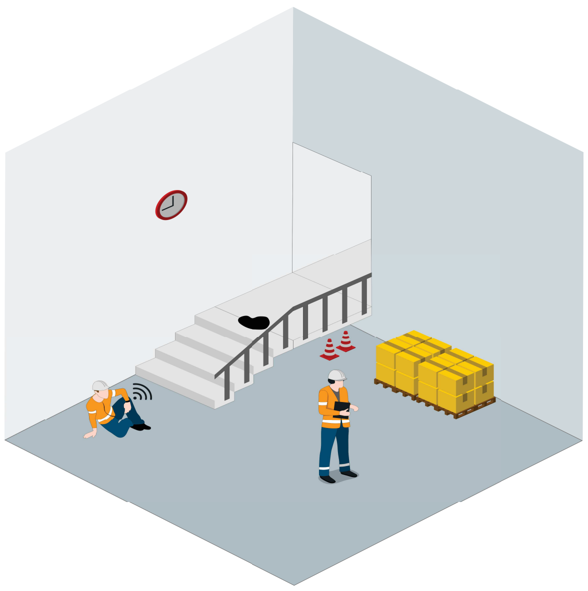
Figure 3.1: Figures show an example of virtual wall application: when an user with *smart glasses* can not entry a specific zone, it will be displayed on them (the figures give only an idea of how this can be achieved); when the user approach the entrance, its access is actually denied. Besides visual notification on glasses, people will be notified also via their smartwatch (if they are wearing one) and their phones.

Simulation This simulation needs two or more users to be performed. One users will be the *man down* person, while the others will embody those who will provide first aid to the injured person. If more than two users will participate, it is possible to choose the best person who can provide first aid basing on the position and the competences. That user will be the chosen one to be notified. The first user will only simulate the *man down* event and wait for help while the others will be assisted in providing first aid via information given by specialized personnel which can not be there immediately. Different events will be tested to show that the *man down* alert works properly and that different event classification is possible. Sensors will provide data to a platform that will select which type of event happened (it can be an injury, a fall, a sickness), helping distinguishing the different kind of emergencies, thus allowing the proper choice of personnel to handle the situation. Once the selected user reaches the *man down* one, he will be helped via tablet, smartphone or smartglasses by being shown interacting visual information on first aid procedure to act.

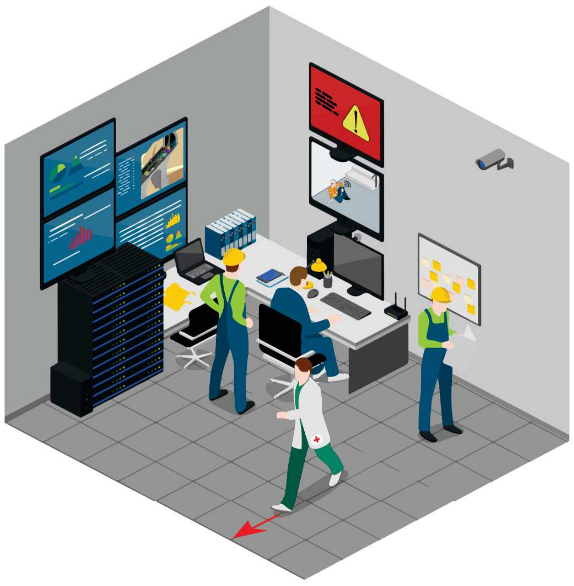
Example: injured user with visible wound A possible scenario involves a man down with a visible wound and a possible bleeding. In this situation the chosen user firstly will receive the proper alert notification, including the kind of *man down* event; after this, he would reach the other user and there, using either tablet, smartphone or smart glasses, connect with remote specialist to show the injury and to be helped providing first aid.



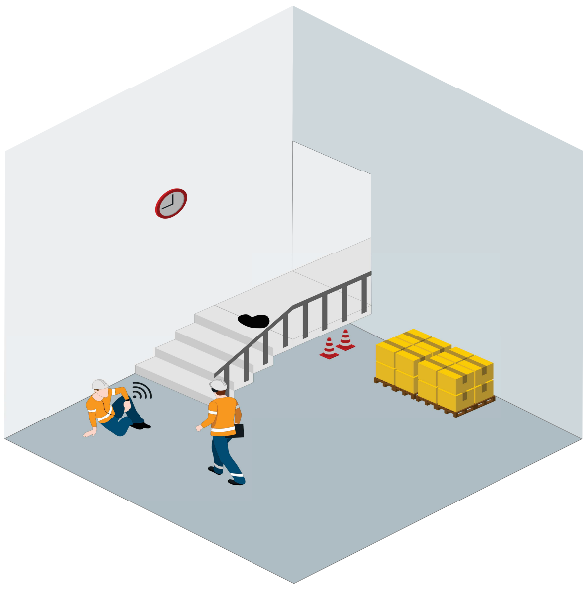
(a)



(b)



(c)



(d)

Figure 3.2: Example of a possible *man down* scenario. In these scenes, a user is having an accindent (a-b) and another busy one is near but did not notice it. The wearable solution sends data to the platform to be elaborated; it generates a *man down* event, which is sent to the control room (c). From there, specialized people can be alerted and notifications can be sent to nearby users (d) so that they can assist the injured one (if they are able to do so) until the proper personnel arrive there.

Chapter 4

Sensor description and analysis

As mentioned before, for this project many different sensors are going to be used. Some of them, such as temperature and gas sensors, come from *Sensor Kit V2.0 for Raspberry Pi B+* [4]. Others, such as BPM detector and accelerometer, are related to *Smartwatch* and are integrated in it [5].

In our case the sensors which we are using are the following:

Temperature sensor: Temperature sensor used in this project is a *thermal resistor*. It is made of semiconductor materials; most thermistors are negative temperature coefficient (NTC) ones, the resistance of which decreases with rising temperature. This effect is due to the increasing number of free charges in the semiconductor thanks to increasing temperature. Since their resistance changes acutely with temperature variations, thermistors are the most sensitive temperature sensors; the precision of NTC thermistors can be up to 0.01 ℃ (depending on the material and how they are built) and the time constant can be less than 10s. A first approximation for their behaviour is $\Delta R = k\Delta T$, with $k < 0$ (NTC), which means that a variation in temperature (ΔT) makes an opposite change in resistance (ΔR). [6]

More precisely, in this specific case the relation between temperature and resistance can be approximated as follow

$$R(T) = R_0 \cdot e^{B[1/(T+273)-1/(T_0+273)]}$$

where $R(T)$ is the resistance measured at temperature T (in Celsius), R_0 is the resistance measured at temperature $T_0 = 25$ °C and B is a coefficient given by the constructor. [7] [8]

This module measures the potential of the thermoresistor with respect to the conventional ground level [7] and then sends it as an analog output which is collected by a Analog/Digital (A/D) converter. It stores the value in its memory and let Raspberry module to go and read it. The code used to collect data converts the analog values into proper temperature values, thus enabling the visualization in a common unit of measure: K, °C, or °F.

Figure 4.1: Thermistor module

Sound sensor: The sound sensor is a component that receives sound waves and converts them into electrical signal. It has a build-in capacitively electret microphone that is sensitive to sound. The electret microphone vibrates with the acoustic wave resulting in the change of capacitance and of the subsequent micro voltage. This voltage is amplified by a operational amplifier and then sent as analog signal to an A/D converter [9] [10]. Due to the way it is built, in this case it is not possible to have a direct conversion of the analog value since it is hard to establish a relation between the sound and the signal due to the membrane vibration.

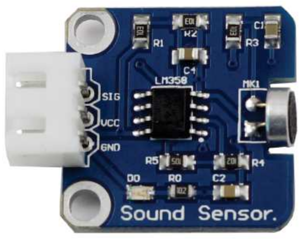


Figure 4.2: Sound sensor module

Gas sensor: The MQ-2 Gas Sensor is a sensor for flammable gas and smoke. It detects the concentration of combustible gas in the air, in particular it is able to recognize liquid petroleum gas (LPG), alcohol, propane, hydrogen, CO and even methane. This kind of sensor is used in gas detecting equipment for smoke and flammable gasses in household, industry or automobile [12].

MQ-2 gas sensor is a kind of surface ion type and N-type semiconductors, which uses tin oxide semiconductor gas sensitive material. It requires a pre-heating period of about two minutes. Tin oxide will adsorb oxygen in the air and form oxygen anion adsorption to decrease electron density in semiconductor so as to increase its resistance. When in contact with the smoke, if grain boundary barrier is modulated by the smoke and changed, it could cause surface conductivity change. In this way, information on the smoke existance can be gained. The higher the smoke concentration is, the more conductive the material becomes, thus the lower the output resistance is, resulting in a higher voltage registered (reflected by a higher analog value). [11]

This gas sensor has both analog and digital outputs. The digital output triggers when a certain gas concentration threshold is passed; it can be tuned via the potentiometer included in the module. The analog output can be sent to A/D converter to be read. The measuments corresponds to the voltage of the surface in arbitrary units. It can be calibrated to get approximately the gas concentration (this passage is not accomplished in this work). [13]

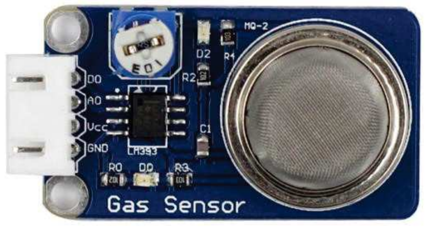


Figure 4.3: Gas Sensor MQ-2 module

Flame sensor: A flame sensor performs detection by capturing infrared photons with specific wavelengths from flame. It can be used to detect and warn of flames. There are several types of flame sensors. In our case, a far-infrared flame sensor is used. It can detect infrared photons with wavelength ranging from 700nm to 1000nm. A far-infrared flame probe converts the strength changes of external infrared light into current changes. It also converts analog quantities into digital ones. This sensor has both analog and digital output as well. Analog values are sent to the A/D converter. Digital output sensitiveness can be tuned by the potentiometer on the module. [14] [15]

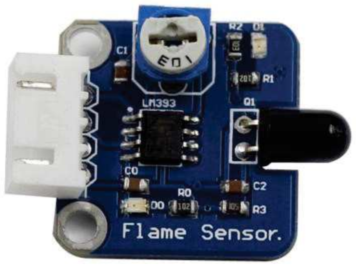


Figure 4.4: Flame sensor

Smartwatch: a smartwatch has its own integrated sensors, which comprise accelerometer, barometer, gyro sensor, HR sensor, light sensor. [5]

The **heart rate sensor** measures heart rate in *Beats per Minute* using an optical LED light source and an LED light sensor. The light shines through the skin, and the amount of light that reflects back is measured. The light reflections will vary as blood pulses under skin past the light. Variations in the light reflections are interpreted as heartbeats. The sensor is located in the device back. [16]

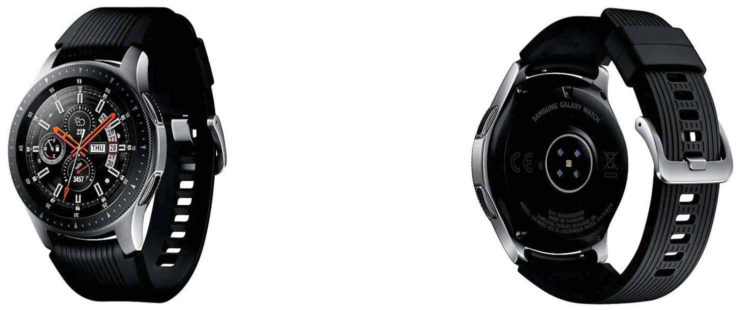


Figure 4.5: Smartwatch (front and back view).

Chapter 5

Data collection and management

In this chapter data collection and management is described. Particular focus will be placed on:

- Data collection and elaboration
- Data flow
- Notifications
- Data historicization
- Data representation
- Data simulation
- Integration platform

Every step in this process is achieved through different applications and stages. Data is collected by the sensors, sent to a server and processed. It can be accessed via a dashboard to monitor real time values.

5.1 Data collection

A data collection process depends on the device data is obtained from. Raspberry and Smartwatch collect and send data in different ways. According to IoT paradigm, they both send data to a centralized elaboration unit via Wifi, LTE or Bluetooth. In this project the focus is on how the Raspberry works and how data collection is implemented. Smartwatch data gathering is mainly automatized and implemented by third party partner company, so it will be described briefly.

5.1.1 Raspberry

Configuration After having downloaded and installed the *RASPBIAN* operating system on a MicroSD, it was configured following the steps of guide [20], (pp. 14 to 21 of the PDF document). To access the device via a desktop, the VNC Service was enabled, following instruction of the aforementioned guide (pp. 30 to 34 of PDF document). Libraries to interact with the hardware were installed as shown in *Appendix A*.

Software-hardware interaction To collect data from the sensors connected to the Raspberry, Python and C scripts can be used. Some examples and code can be downloaded from the SunFounder web page [24]. Python scripts were chosen for their syntax simplicity. An initial analysis was carried out to understand the example scripts and how the sensors interact with the raspberry. As a result, a script collecting data from multiple sensors at the same time was implemented. Sensors used for data collection are: temperature sensor (thermistor), sound sensor, gas sensor, flame sensor.

Sensors-Raspberry interaction The Raspberry module collects data from the sensors connected to it. As described before, there are different sensors that collect distinct environment data. They communicate with Raspberry module in two main ways:

- directly, as Boolean signals (high-low), connecting to Raspberry pins;
- via an A/D converter, which stores the data in its registers and then is used as a slave by the raspberry module in a ip-bus framework

The A/D converter is a PCF8591 module. It has four analog inputs, one analog output and a serial I2C-bus interface. Address, control and data to and from the device are transferred serially via the two-line bidirectional I2C-bus. [17]

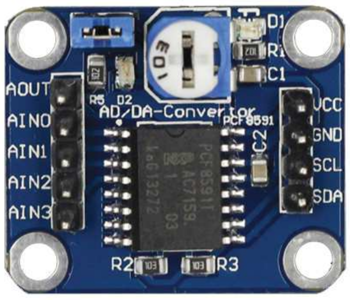


Figure 5.1: PCF8591 8-bit A/D and D/A converter Module.

Different modules are connected to different A/D channels of the PCF8591 converter and send their information to it. Raspberry reads the converter's registers to collect the data (to read the registers, a proper Python library is used; it can be found on the vendor's web page, together with the code examples). The code for the data collection can be found in the *Appendix B*.

Every sensor was tested in order to prove it can detect proper environment information and the results are reported in **Appendix C**. As explained before, the analog data unit of measure are arbitrary since is not always possible to convert them from stored analog value to a proper number describing something specific (e.g. gas density). For the **flame sensor** a lighter was ignited in front of it and moved to different distances. Changes in the value registered by the sensor correspond to different distances of the flame with respect to the sensor (the relation is inversely proportional due to the way sensor is build). For the **temperature sensor**, it was left on a surface for a while, then kept in contact with a hand: the temperature grew quickly when touching it and decreased back when the hand was taken away. For **MQ-2 gas sensor** a lighter was used to release some gas nearby it (without igniting it): detected value increased as the gas density increased and went back to previous plateau value when gas was blowed away. For what concern **sound sensor**, some noise was made to test if it could catch it. Since the way it detects the membrane vibration are instantaneous, sometimes its status is registered as "normal" (no noise) also if actually some noise was made (this can be to the fact that in the exact instant the measure was taken, the sensor was not vibrating that much). This was not considered an important issue because in the case there would be an unbearable noise, the module would detect it anyway.

5.1.2 Smartwatch

Smartwatch data is sent to a third-party platform. Its task is to gather data from the accelerometer, altitude and HR sensors [18]. In this platform an algorithm takes care of carrying out a first analysis to look for correlations in the data via a *man down* events detection patterns. These kind of events can be represented by a user feeling sick (e.g. lying on the ground for a long time and/or having very low or very high heart rate), stopping for a long time (using GPS sensors), falling from a serious height (using altitude and accelerometer sensors), suffering dangerous impacts. In all of these cases, the algorithm recognizes if the user can be in a dangerous situation and it sends a predefined type of data and notifications to the integration platform.

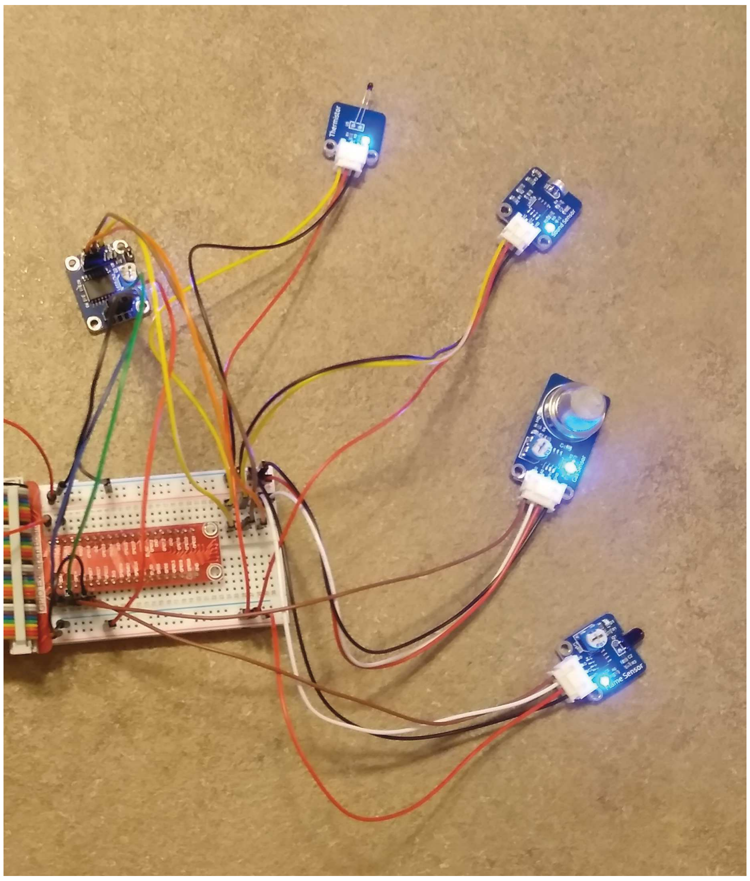


Figure 5.2: From left to right and from top to bottom: PCF8591 module, thermistor module, sound sensor module, gas sensor module, flame sensor module.

5.1.3 Data collection in real situations

It is clearny not possible to collect data about real fires, gas leaks or dangerous accident implementing simulations involving human beings. This kind of test can be made placing the sensors on a small movable object or simply placing them near a real restrained danger (e.g. a controlled gas leak or fire); a dummy can be used to make some *man down* tests which mimic dangerous accelerations or falling from high places. These tests should be done not only to check the sensors functioning but also to check that they can be used in real life scenarios and that the whole system concerning data collection and integration works well.

5.2 Data flow

Data is meant to be sent to an *integration platform* whose task is to correlate the data and to eventually send notifications about possible dangerous situations to the dashboard, which should be kept under observation in a control room. This platform is a server which task consist in gathering information from all sources; it collects and elaborate raspberry data, while it only collects smartwatch data coming from the third-party platform and not directly from the smartwatch. (Figure 5.3).

To transfer data from the raspberry module to the platform a Node-RED [25] flow is used (Figure5.4). It brings data from the file generated by the Python script and insert the different sensors values in a JSON formatted message; this format was chosen because it is a standard format and can be easily parsed by most software. The created message is sent to the integration platform. In **Appendix D** the code of the *transformToJSONFormat* node is reported, showing the logic to collect the data and generate a message containing all information the integration platform needs to identify the message, such as the device which is sending them, the position, etc.

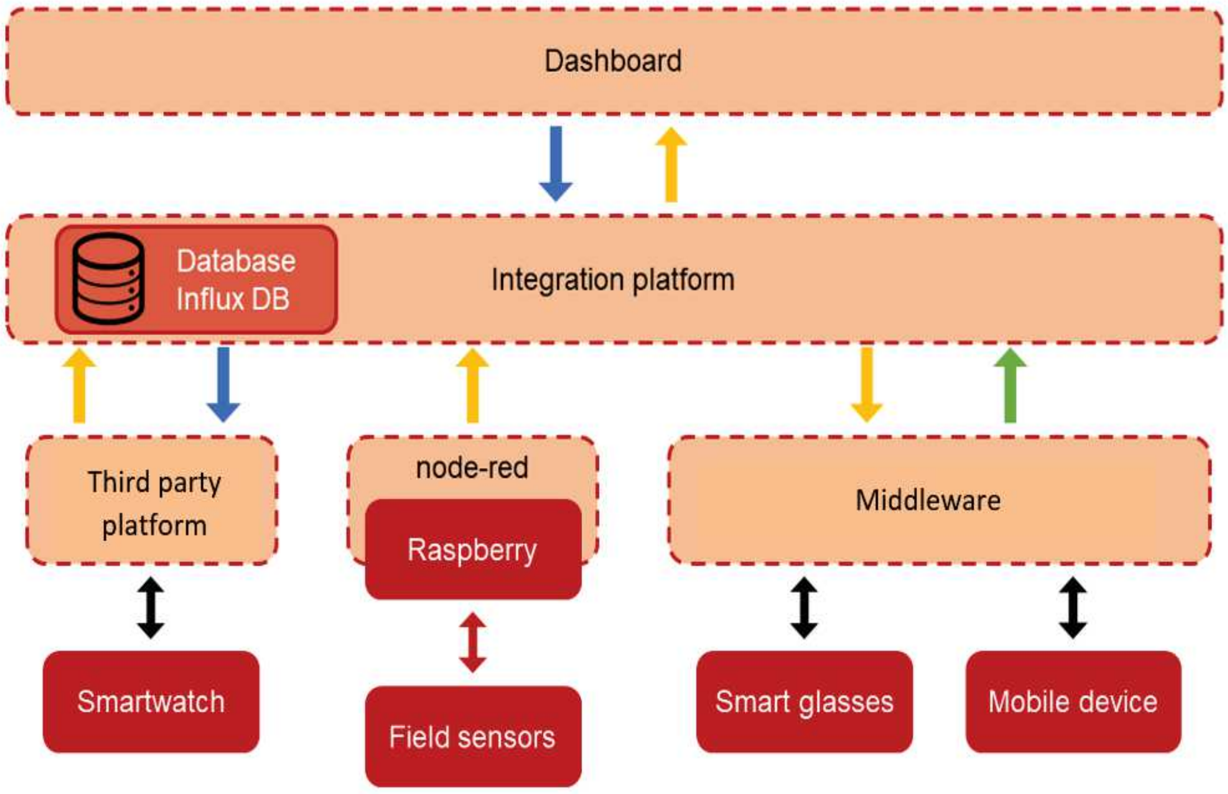


Figure 5.3: Schema of the different section of the whole architecture. Smartwatch sends data to a third party platform that elaborates it and send proper messages to the integration platform. Sensors collect data and communicate with the raspberry module; after a brief analysis, it sends the data to the integration platform. The integration platform takes care of storing data and performing deeper analysis on it. It communicates with the real time dashboard and with the smart objects the user may has, such as smart glasses and a mobile device.

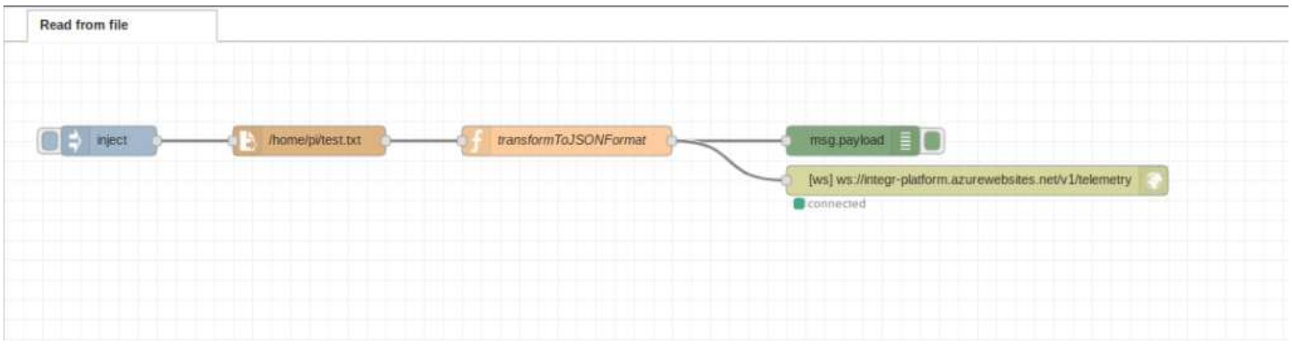


Figure 5.4: Figure shows Node-Red flow to collect data from file generated by python script (conventionally called "test.txt"). Data is sent to a debug node and to the integration platform via *web socket* packages.

5.3 Notifications

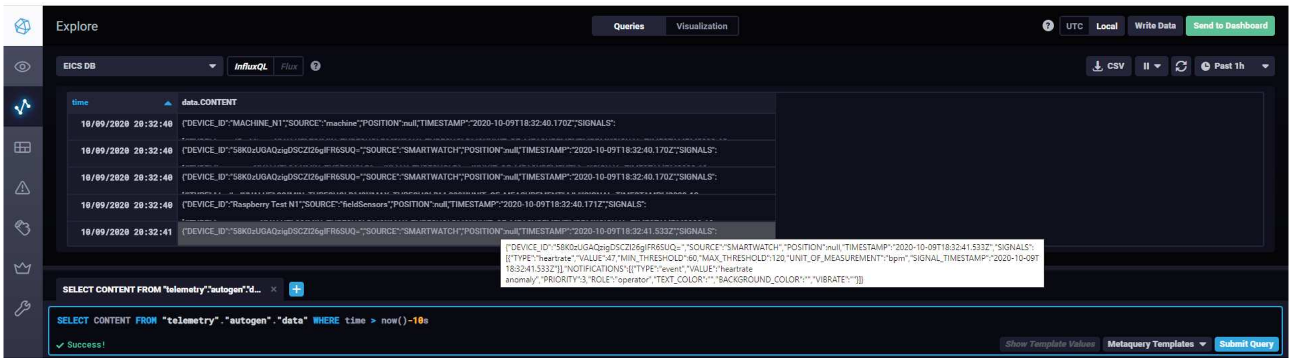
Integration platform verify if the (raspberry) data have anomalous values. In case of dangerous values (e.g. density of flammable gas is too high or in case of fire detection) it generates and send notifications to the dashboard and to users which are facing the risk.

In case of a *man-down* event, the integration platform takes care of sending the notification to the nearest proper user, if any (e.g. to nearest first aid qualified users).

5.4 Data storing

Data collected by the integration platform is saved into a database for further analysis. In this scenario, an algorithm reads and elaborate the data in order to verify anomalous situations which are not instantaneously detectable; e.g. when a user is laying down for a long time interval (gyroscope and altitude data), without moving (accelerometer data) and maybe a user is having some health problems (irregular pattern in heart rate detection).

In all the aforementioned cases, a notification is generated by the integration platform basing on the event that occurred.



time	data.CONTENT
10/09/2020 20:32:40	{"DEVICE_ID":"MACHINE_N1","SOURCE":"machine","POSITION":null,"TIMESTAMP":"2020-10-09T18:32:40.170Z","SIGNALS":
10/09/2020 20:32:40	{"DEVICE_ID":"58K0zUGAQzigDSCZI26gIFR6SUQ=","SOURCE":"SMARTWATCH","POSITION":null,"TIMESTAMP":"2020-10-09T18:32:40.170Z","SIGNALS":
10/09/2020 20:32:40	{"DEVICE_ID":"58K0zUGAQzigDSCZI26gIFR6SUQ=","SOURCE":"SMARTWATCH","POSITION":null,"TIMESTAMP":"2020-10-09T18:32:40.170Z","SIGNALS":
10/09/2020 20:32:40	{"DEVICE_ID":"Raspberry Test N1","SOURCE":"fieldSensors","POSITION":null,"TIMESTAMP":"2020-10-09T18:32:40.171Z","SIGNALS":
10/09/2020 20:32:41	{"DEVICE_ID":"58K0zUGAQzigDSCZI26gIFR6SUQ=","SOURCE":"SMARTWATCH","POSITION":null,"TIMESTAMP":"2020-10-09T18:32:41.533Z","SIGNALS":

{"DEVICE_ID":"58K0zUGAQzigDSCZI26gIFR6SUQ=","SOURCE":"SMARTWATCH","POSITION":null,"TIMESTAMP":"2020-10-09T18:32:41.533Z","SIGNALS":
[{"TYPE":"heartrate","VALUE":47,"MIN_THRESHOLD":60,"MAX_THRESHOLD":120,"UNIT_OF_MEASUREMENT":"bpm","SIGNAL_TIMESTAMP":"2020-10-09T
18:32:41.533Z"}],"NOTIFICATIONS":[{"TYPE":"event","VALUE":"heartrate
anomaly","PRIORITY":3,"ROLE":"operator","TEXT_COLOR":"","BACKGROUND_COLOR":"","VIBRATE":""}]}

SELECT CONTENT FROM "telemetry"."autogen"."d...

```
SELECT CONTENT FROM "telemetry"."autogen"."data" WHERE time > now()-10s
✓ Success!
```

Figure 5.5: Figure shows the database in which data are stored. A query example is operated and a piece of data is shown completely. It can be noticed the JSON format used to send and store the data.

5.5 Data representation

As explained before, data can be used to monitor the state of the users by some *Security* personnel. For this purpose, a dashboard has been developed (Figure 5.6). It allows the real time monitoring of the positions and the main parameters of the users, besides the visualization of recent past data via some graphs. This way it can be used by people in a *control room* to monitor the users and eventually manage emergencies faster.

5.6 Data simulation

As already mentioned, it is very hard to collect data from real situations as those aforementioned. To test integration platform, data collection, data historicization and notification dispatching, data in non-danger situations were collected and some simulations were made in order to generate data for more risky events; plausible data were generated via Node-red flows. It is then formatted via some functions and routed to the typical data flow concerning data historicization and elaboration.

5.7 Integration platform

The integration platform is the most important component of the system. Its task is to collect data from all sources and elaborate them to eventually send notifications to the control room and to

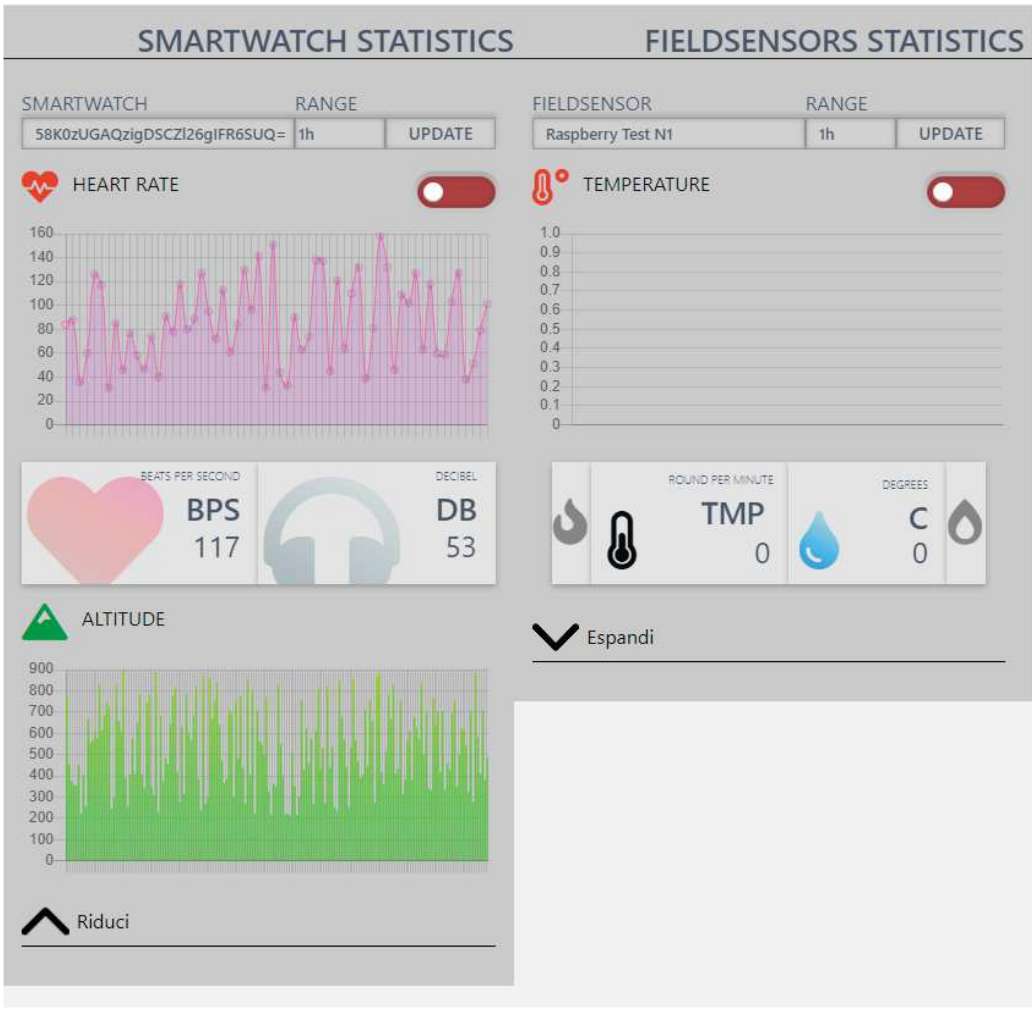
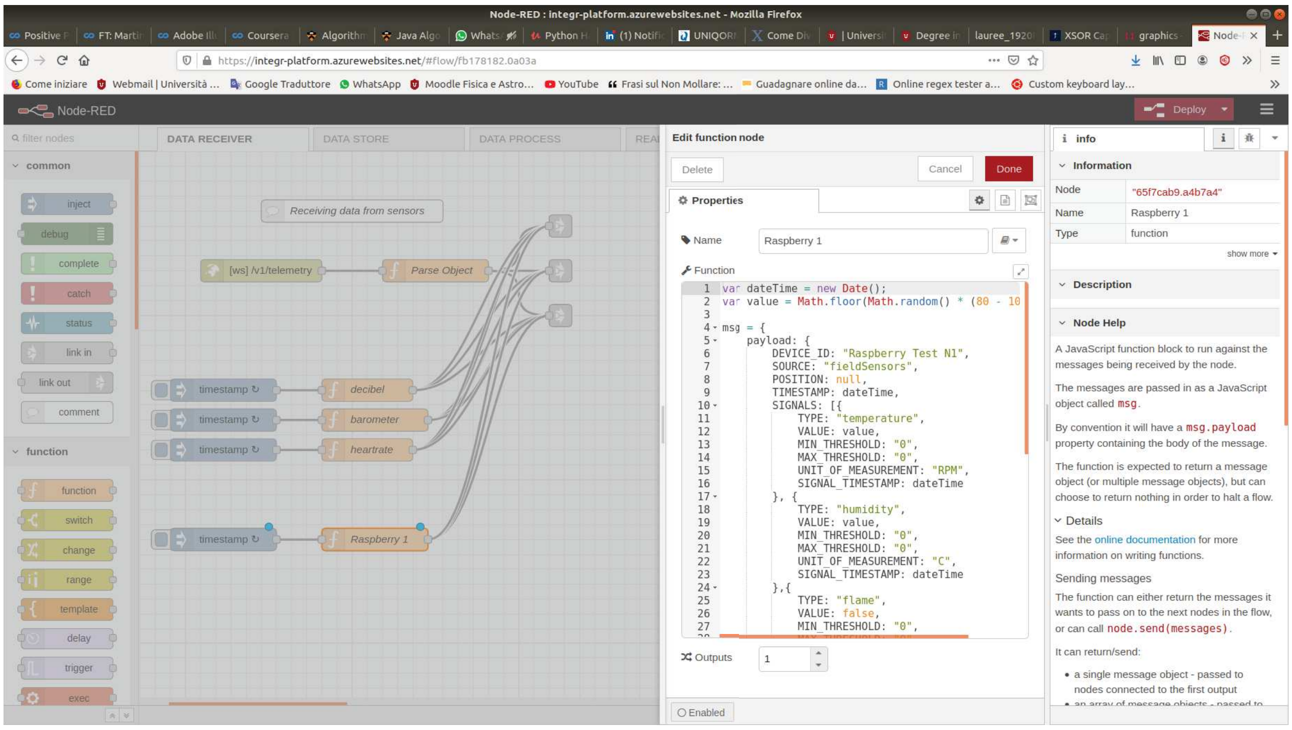


Figure 5.6: Figure shows a prototype of the dashboard. As mentioned, it can shows real time parameters as well as graphics showing data in a selected interval of time. In this example, smartwatch test data is loaded from database.

users. To test it, some situations were performed and some other simulation data concerning more dangerous events were generated.

Data from all possible sources is generated via the *data receiver* tab (Figure 5.7a), then it is sent to three different stages. The first stage involves data storage (Figure 5.7b); here data is sent to a database to be registered and be accessible for deeper analysis. Second step involves data analysis (Figure 5.7c): here some first correlations are checked analyzing if data about flammable gas and flames are both present in a short period of time or if there is a *man-down* event and a high variation in altitude sensor measures. Both this events imply a more dangerous situation with respect to that represented by those signals alone. In these cases, more serious event are generated and sent both to database and to final stage integration platform. The third stage represents the core of the integration platform (Figure 5.7d), collecting data from all sources and sending back notifications to dashboard and to the user. The last tab is a simulation of queries sent to the database to retrieve data (Figure 5.7e); data storing is important because it allows to look for special situations that are characterized by time dependent events, such as the falling from a high linked to a *man-down* event, which can not be spotted by instantaneous data.



```
var dateTime = new Date();
var value = Math.floor(Math.random() * (80 - 10

msg = {
    payload: {
        DEVICE_ID: "Raspberry Test N1",
        SOURCE: "fieldSensors",
        POSITION: null,
        TIMESTAMP: dateTime,
        SIGNALS: [{
            TYPE: "temperature",
            VALUE: value,
            MIN_THRESHOLD: "0",
            MAX_THRESHOLD: "0",
            UNIT_OF_MEASUREMENT: "RPM",
            SIGNAL_TIMESTAMP: dateTime
        }, {
            TYPE: "humidity",
            VALUE: value,
            MIN_THRESHOLD: "0",
            MAX_THRESHOLD: "0",
            UNIT_OF_MEASUREMENT: "C",
            SIGNAL_TIMESTAMP: dateTime
        },{
            TYPE: "flame",
            VALUE: false,
            MIN_THRESHOLD: "0",
```

Information

Node	"65f7cab9.a4b7a4"
Name	Raspberry 1
Type	function

Description

Node Help

A JavaScript function block to run against the messages being received by the node.

The messages are passed in as a JavaScript object called `msg`.

By convention it will have a `msg.payload` property containing the body of the message.

The function is expected to return a message object (or multiple message objects), but can choose to return nothing in order to halt a flow.

Details

See the online documentation for more information on writing functions.

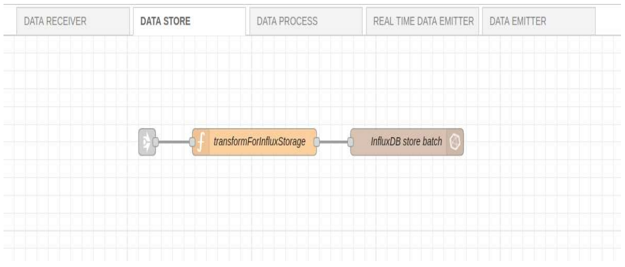
Sending messages

The function can either return the messages it wants to pass on to the next nodes in the flow, or can call `node.send(messages)`.

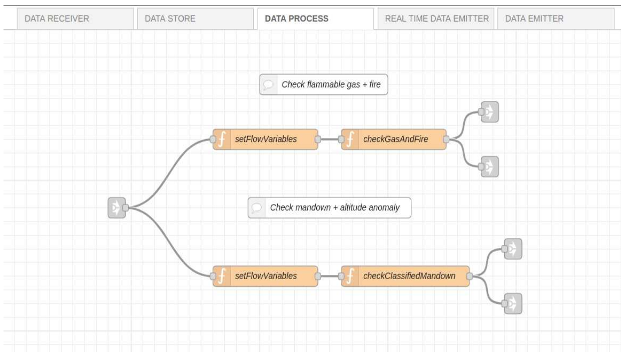
It can return/send:

- a single message object - passed to nodes connected to the first output

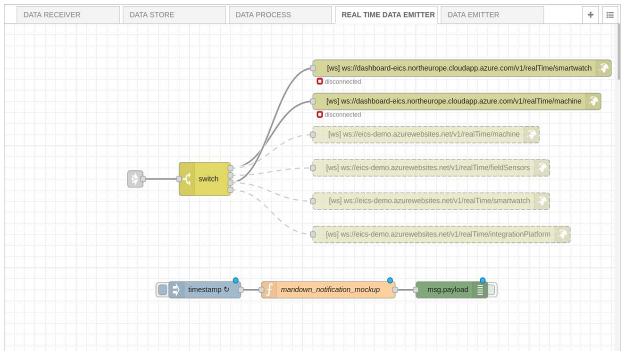
(a)



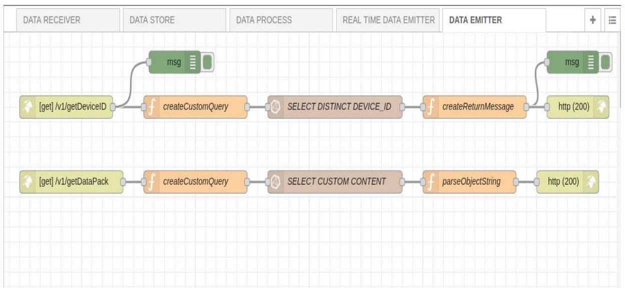
(b)



(c)


DATA RECEIVER
DATA STORE
DATA PROCESS
REAL TIME DATA EMITTER
DATA EMITTER
[ws] ws://dashboard-eics.northeurope.cloudapp.azure.com/v1/realTime/smartwatch
disconnected
[ws] ws://dashboard-eics.northeurope.cloudapp.azure.com/v1/realTime/machine
disconnected
[ws] ws://eics-demo.azurewebsites.net/v1/realTime/machine
[ws] ws://eics-demo.azurewebsites.net/v1/realTime/fieldSensors
[ws] ws://eics-demo.azurewebsites.net/v1/realTime/smartwatch
[ws] ws://eics-demo.azurewebsites.net/v1/realTime/integrationPlatform
switch
timestamp ↻
mandown_notification_mockup
msg.payload

(d)



(e)

Figure 5.7: Figures show the integration platform flows, based on Node-red. The first one (a) shows the tab in which data is collected from real sensors and eventually generated. A node is expanded to show how the code is structured; every other node has a similar structure. The others shows respectively the tabs in which data is sent to the database (b), where data is elaborated looking for correlations (c), where event and notification management is carried out (d) and finally the tab in which some example queries are generated to test the database response.

Chapter 6

Data and algorithm analysis

Data from real scenarios simulations and generated data was collected and analyzed. Simulated data was built *ad-hoc* trying to best simulate real scenarios to check algorithm responses.

This data was used to test a first version of the algorithm to check if it can detect most obvious evidence of dangers and to see if it can distinguish a probable accident with respect to a sensors malfunction.

At a first glance, data that can be correlated and which are easier to collect or simulate are:

- Heart rate and sound values
- difference in altitude
- Presence of both flames and gas in a short period of time

All these kind of relations can be evidence of some particular event that can turn out to be serious. All of them should be captured as soon as possible. For this purpose, a balance is needed between data collection frequency (that can arrive to arbitrary frequencies, e.g. 10 times per second), the network usage of every device (if every device is always sending data, the network may not be able to handle much traffic), the storage capability of the system (collecting much data implies higher memory usage, which can have a cost and can impact the time during which data is retrievable). Simulations were done using only one device; however to have a realistic analysis, data was chosen to be collected every ten seconds, thus allowing a more realistic scenario.

6.1 Heart rate and sound values

A first analysis was carried out using data coming from easier to simulate events. They comprise common situations, which involves less risk for the user but which are nevertheless useful to the integration platform to detect possible dangers for the human being, especially if they are not immediate and their consequences raise due to a prolonged exposure. To begin with, heart rate and sound data were collected. For this goal, a ten minute simulation was made; some noise was produced at times (hitting the table and speaking loud) near a user wearing the sensors; in that period, both sound and heartbeat measures were recorded. All the other parameters were kept as constant as possible; the user was kept in the same room (so the temperature was considered constant and not taken into account), he was sat and thus he could be considered at rest; no flame nor gasses were present in the room. Results are reported in (Figure 6.1 and 6.2).

Comparing the results (Figure 6.3), highlighting the periods in which there is more noise, there seems to be a correlation between the presence of noise and the increased heart rate. To try to give en estimation of it, conventionally we assigned values "0" for a quiet situation and "1" for noise.

Evaluating the correlation coefficient between the series, using the formula $r_{xy} = \frac{\sum_i((x_i-\bar{x})\cdot(y_i-\bar{y}))}{\sqrt{((\sum_i x_i-\bar{x})^2)\cdot(\sum_i y_i-\bar{y})^2}}$, the values obtained are reported in Table 6.1.

These values indicates poor correlation between the noise and the heart beat ratio in the two samples, but nevertheless it seems that there is a considerable influence. With a better insight on the

	First sample	**Second sample**
r	0.500	0.557

Table 6.1

cases, there are many factors that can lead to uncertainty about this preliminary result. First, we must take into consideration the fact that the values of noise are only boolean ones: this fact can lead to a less rich noise sample, constraining the measures with respect to a threshold and leading to poor results. The threshold itself can be tuned (in the two data acquisitions the sound sensor was used without modifying the threshold value), thus leading to slightly different results. Even if the most of the distractions were removed while taking measures, the heart rate can be influenced also by other factors, such as the some unexpected event, some shock, how relaxed the user is, or the thoughts that come to his mind.

To improve this kind of measure, the data can be collected in a more controlled situation or even better, in real situations while testing the solution on real users during their activity, even if in that case there will be more factors participating and influencing each other, such as physical effort (due to work), noise, temperature, stress. Furthermore, the sensor can be changed using one that can take a proper sound measure in db instead of measuring a threshold for noise.

Finally, even if this type of correlation seems quite obvious, in some studies doubts are expressed about it [27].

This kind of analysis is carried out because it can be valuable as first step to study the time a user is exposed to noise during a workday and the effects this has on his health [27].

6.2 Altitude simulations

To test the integration platform algorithm detecting altitude anomalies some data simulations were built *ad-hoc*. The main goal of the algorithm is to detect if there are some serious variations in the altitude value: in particular, these differences should be smaller than 0, thus indicating a possible fall; they should be of a consistent magnitude, otherwise also a user getting down to lie his shoes would be detected as a particular event, the altitude being measured by a smartwatch; they should be "instantaneous", they should not persist in time since this situation can be a signal that someone is, for example, going downstairs. In simulations, values are stored every 10 seconds. Three simulations were made in order to try different scenarios: in the first one (Figure 6.4), a user going upstairs is simulated, adding about 3m to the baseline value in 10 seconds; in the second one (Figure 6.5), a possible fall event is simulated, involving a decrease in altitude of about 3 meters in two successive points of the series (thus in less than 10 seconds); in the third one (Figure 6.6), a user going downstairs two floors is simulated, making the changes in the measures lasts for longer. If the algorithm spots an anomaly in the altitude values, it looks for the presence of *man down* events sent by the third party platform to the integration platform and regarding the same user. If it finds one, they are treated as a confirmation of each other and a more serious notification is sent to the dashboard. If there are no *man down* events, the algorithm goes on and waits for latest data.

In the following, data simulations and their results in the form of graphical representations are showed; if the algorithm did not trigger, nothing but the altitude records are shown in the figure.

6.3 Gas and Flames simulation

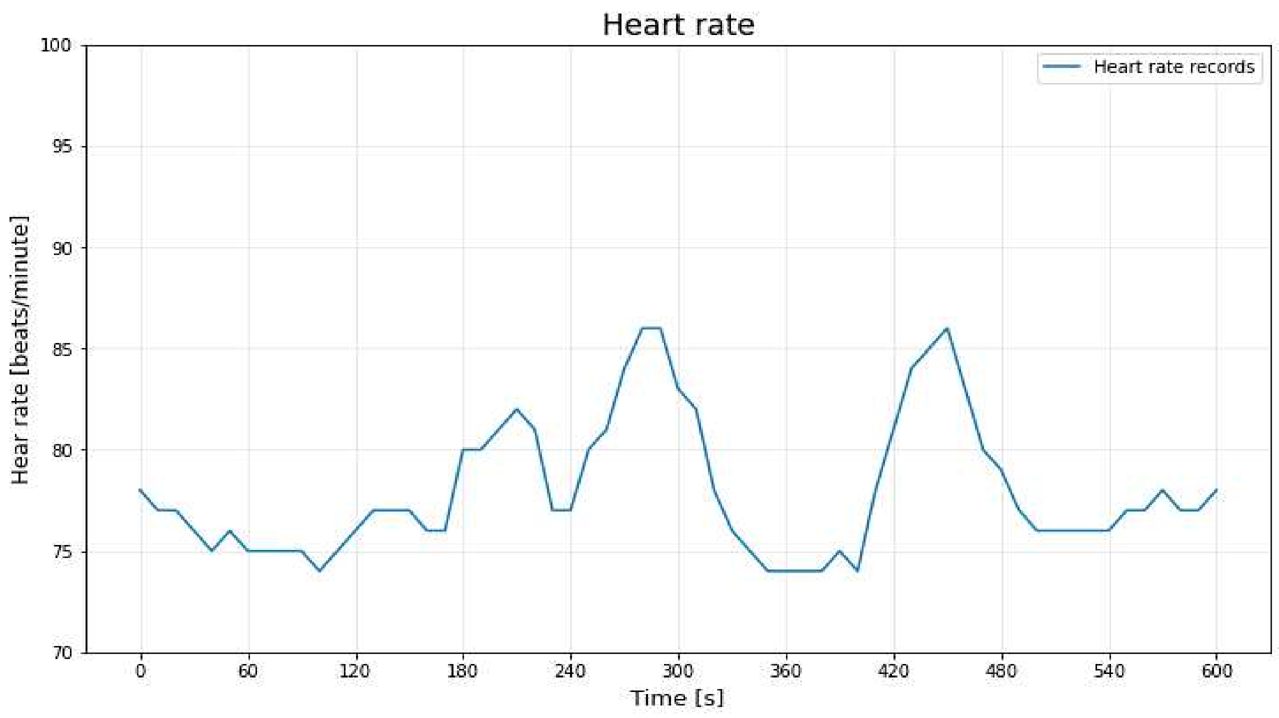
For gas and flame data some simulations were made in order to verify the correct functioning of the algorithm. When a gas or flame event is detected, the algorithm looks for other events of the other to eventually notify that there are two dangers which can possibly combine to give a more serious one. To simulate them, some random events were simulated in a 10 minutes period. When the algorithm goes through the series, it checks the events in the following way:

- if a flame event is detected, gas events since the previous 30 seconds are also checked;

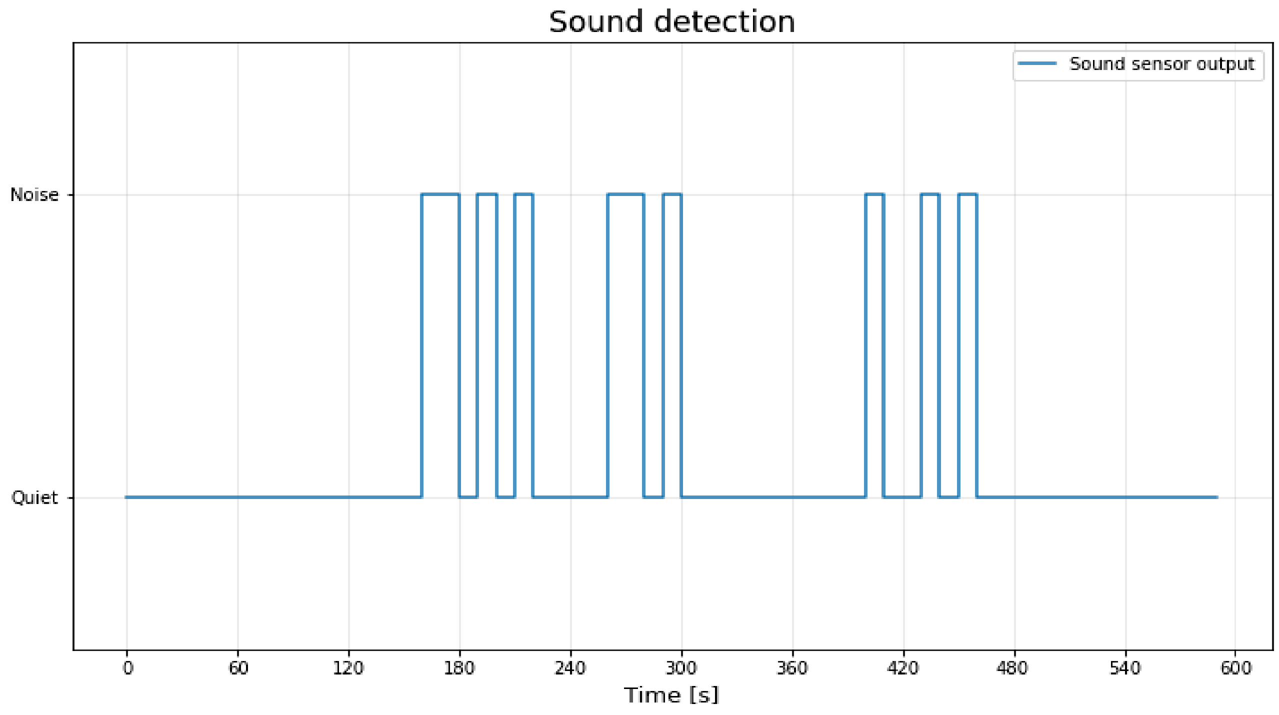
- if a gas event is detected, flame events since the previous 10 seconds are also checked.

This difference in the verification are due to the different nature of the considered dangers: gas is more prone to stay in a zone when is detected, while if flame detection stops, than probably there are no more flame in the area, but a check is made anyway on the previous period.

Simulations results are reported below. In the figures, events ”tails” represent the time windows in which the algorithm looks for the events of the other kind. In the first simulation no event combinations were found. In the second one, two events occurred simultaneously and the algorithm responded well, generating a more important notification and sending it to the dashboard.

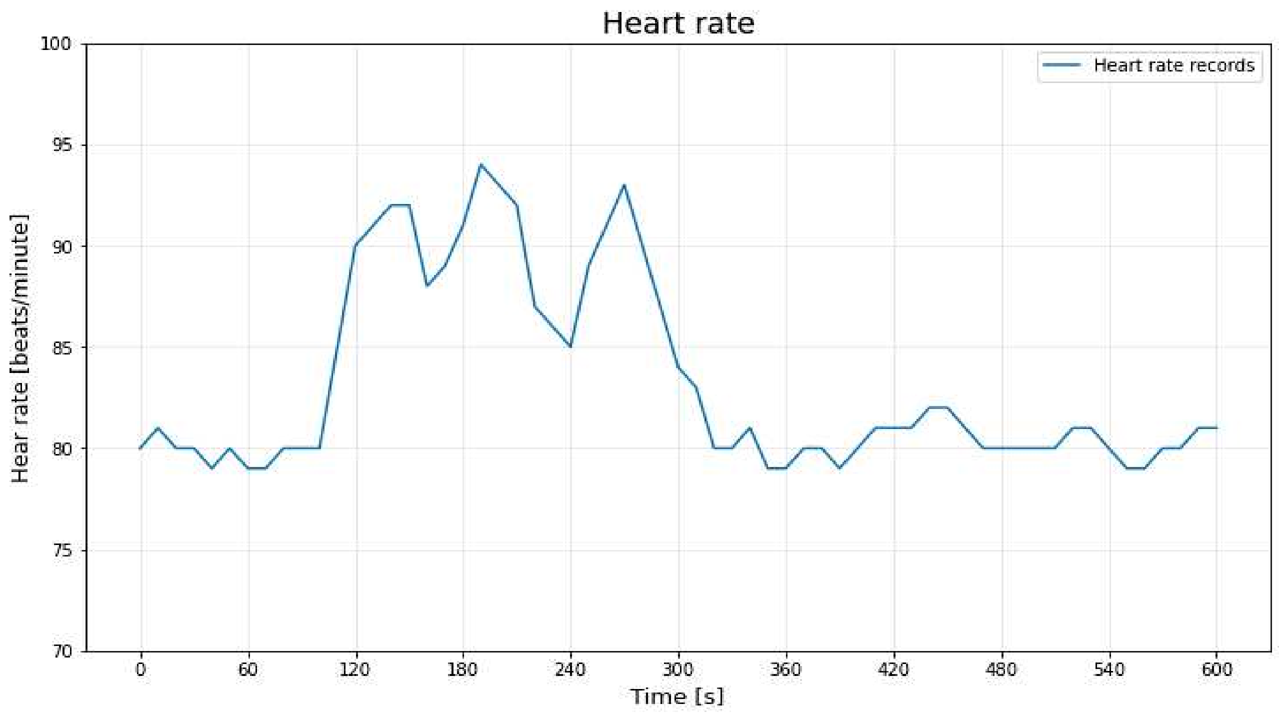


(a)

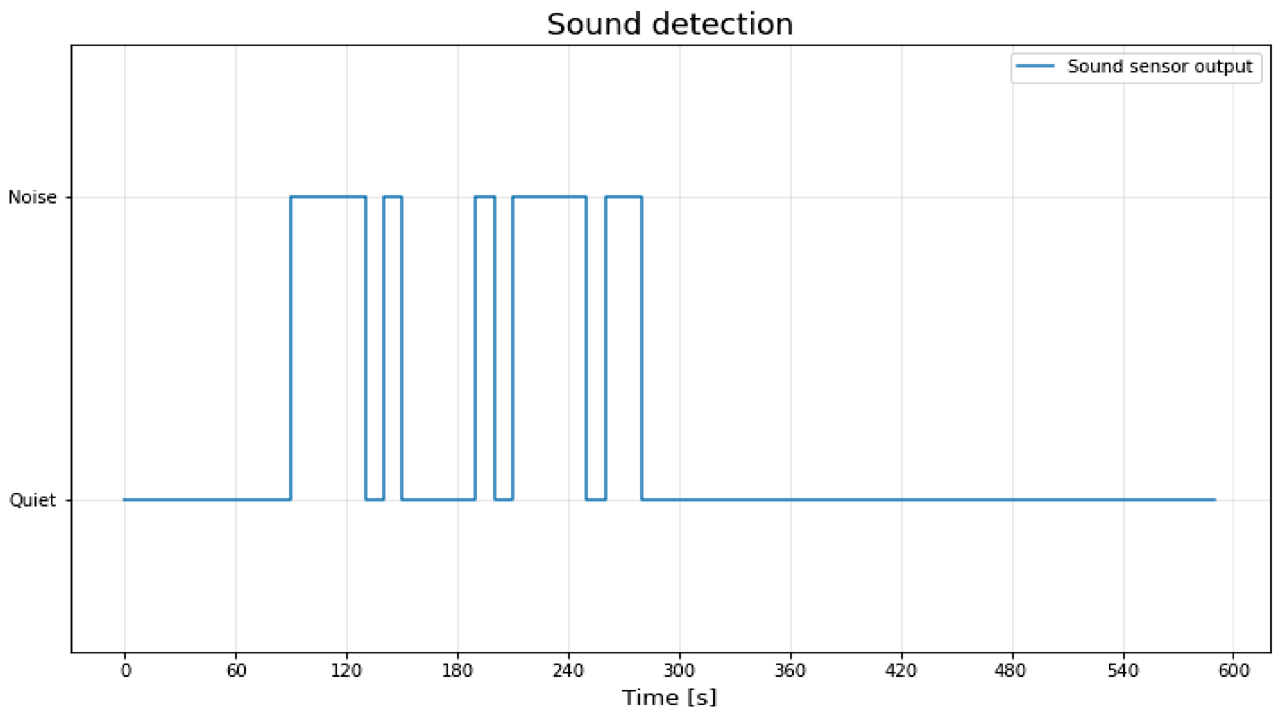


(b)

Figure 6.1: Figures show data collected in a 10 minutes period for noise sensor and heart rate. Figure (a) shows heart beat measures with respect to time, while figure (b) shows the values displayed by the sound sensor (remember the sound sensor used claims there is noise evaluating if its signals exeed a threshold).

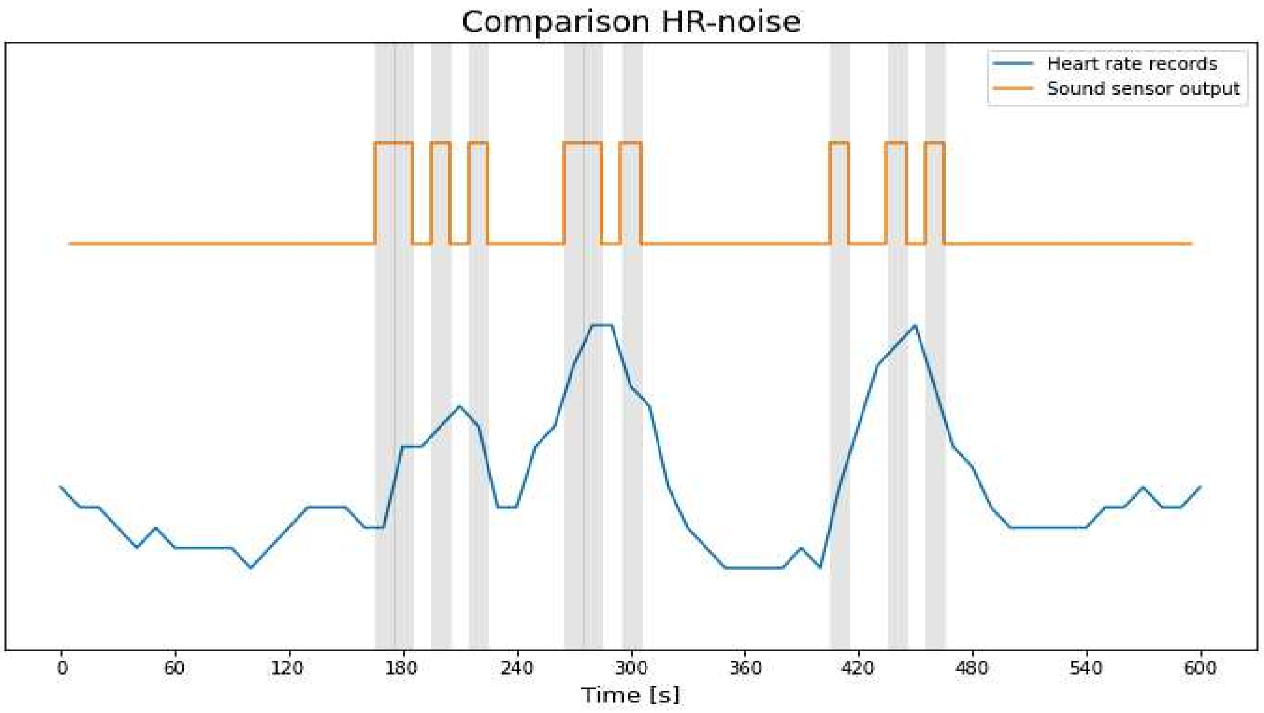


(a)



(b)

Figure 6.2: These figures show other data for noise sensor and heart rate. Figure (a) shows heart beat measures with respect to time, while figure (b) shows the values displayed by the sound sensor.



(a)

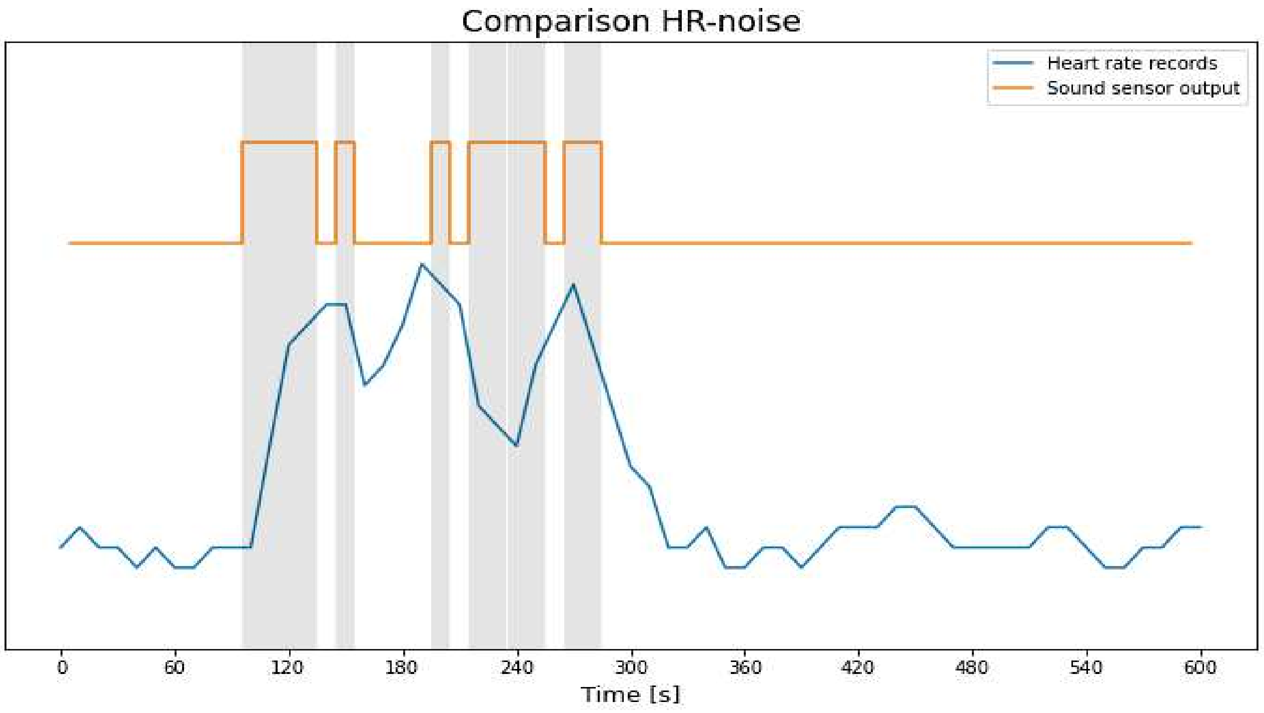


Figure 6.3: Figures show time comparison between sound sensor data and heart rate data. Areas in which the sound is detected are highlighted to have a clearer comparison. The two series actually seem to be correlated; when there is noise, the user's heart rate seem to be higher.

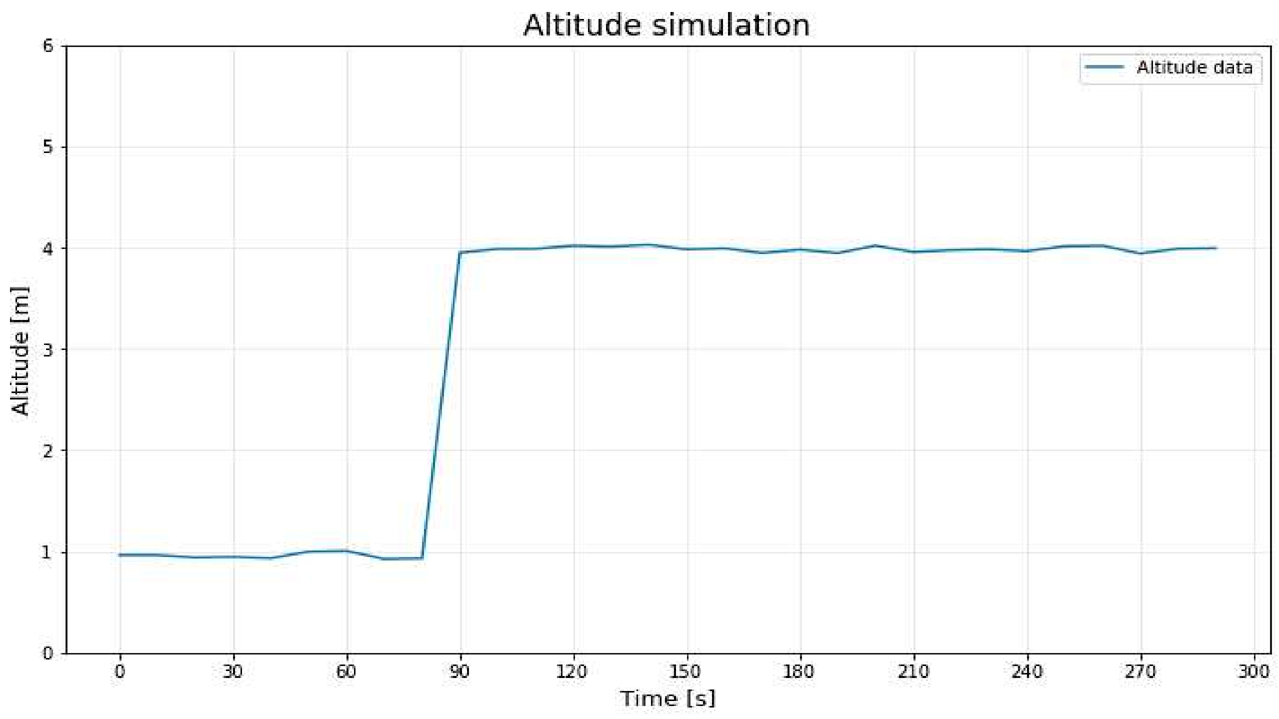


Figure 6.4: Figure shows a simulation of altitude measure of a man going upstairs. In this case the algorithm should not detect dangers (and indeed it does not) because there is no clue of a dangerous situation.

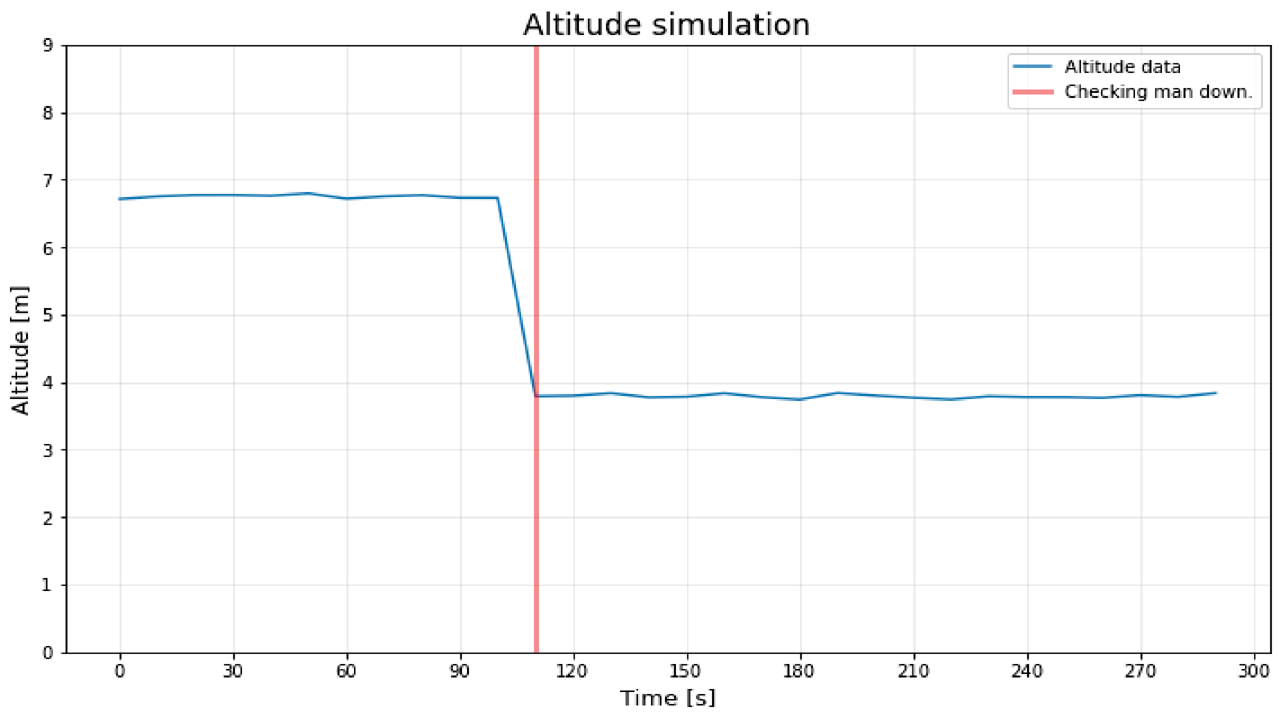


Figure 6.5: Figure shows second kind of simulation in which there is a possible fall event. In this case, the integration platform algorithm checks for *man down* events notification from the third-party platform; if the check is successful, a more serious notification is sent to the dashboard and thus to the control room.

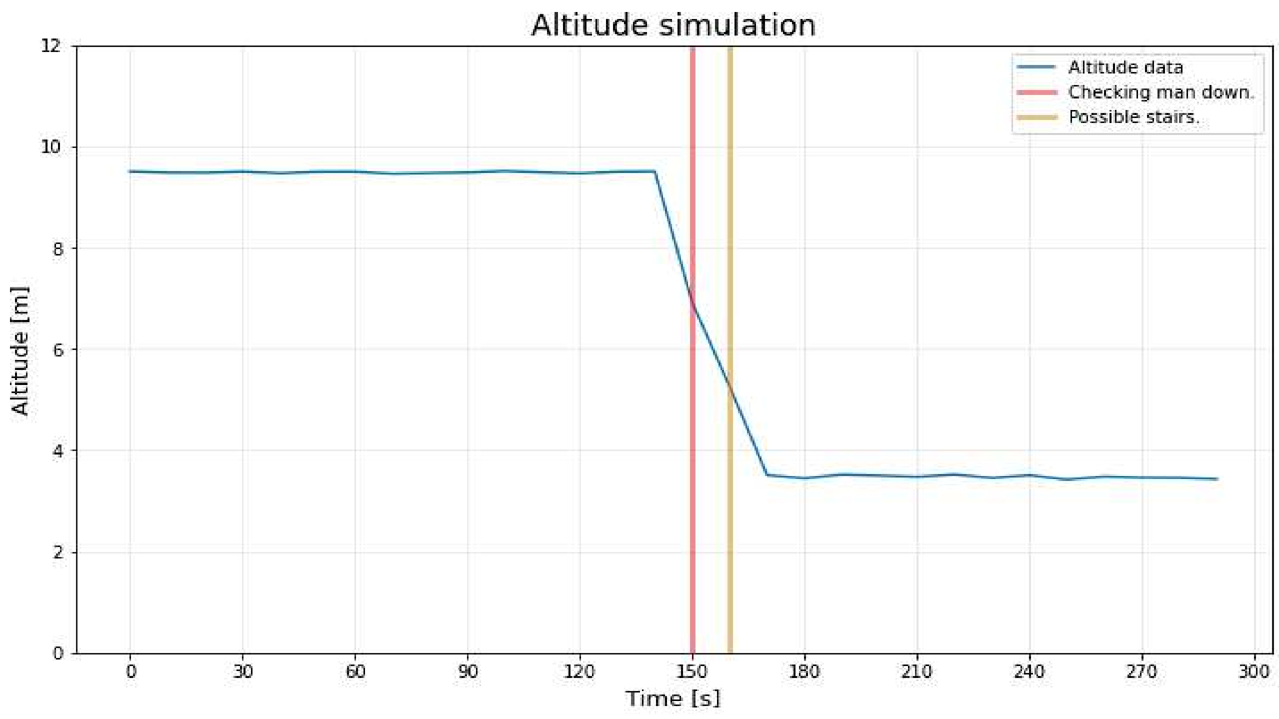


Figure 6.6: Figure shows a case that can be interpreted as a user going downstairs. While analyzing the data, the algorithm recognize that there is the possibility of an accident and starts checking for *man down* events; in the successive step, it recognizes that probably there is no serious danger and it keep going on.

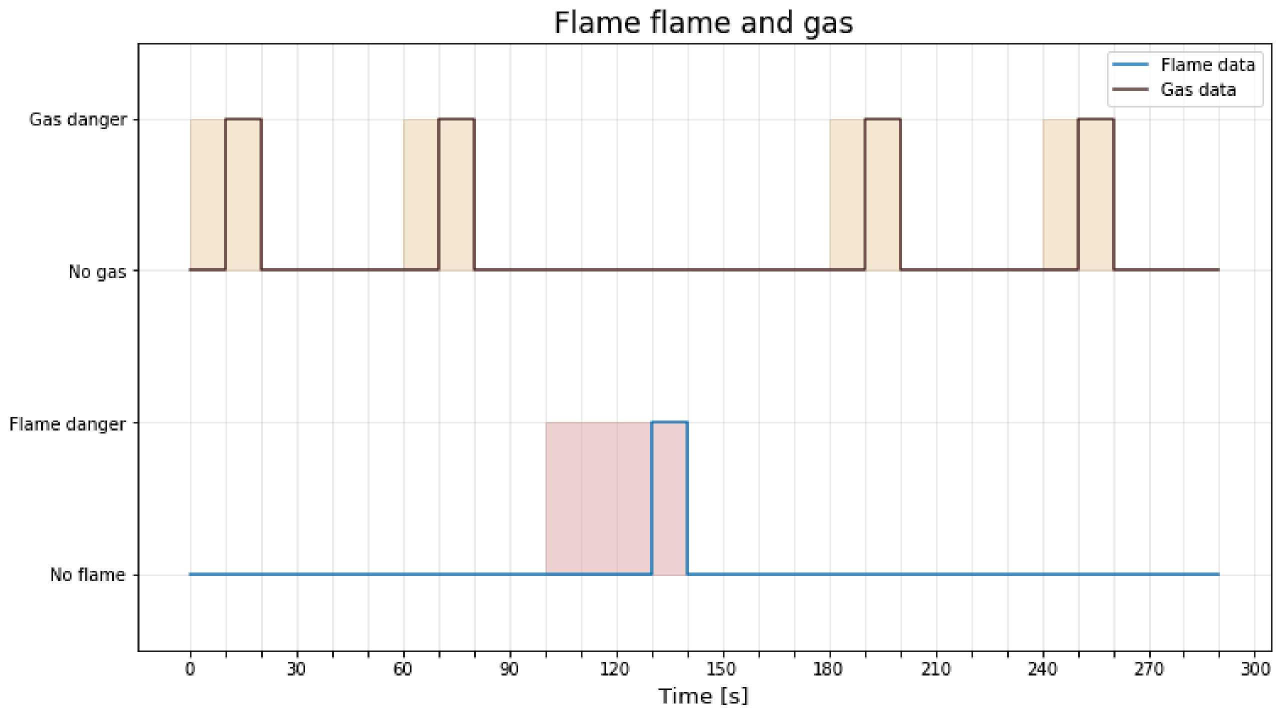


Figure 6.7: Figure shows the data of a simulation of gas and flame sensors. In this case there are no combined events and the tails never include a signal of the other kind.

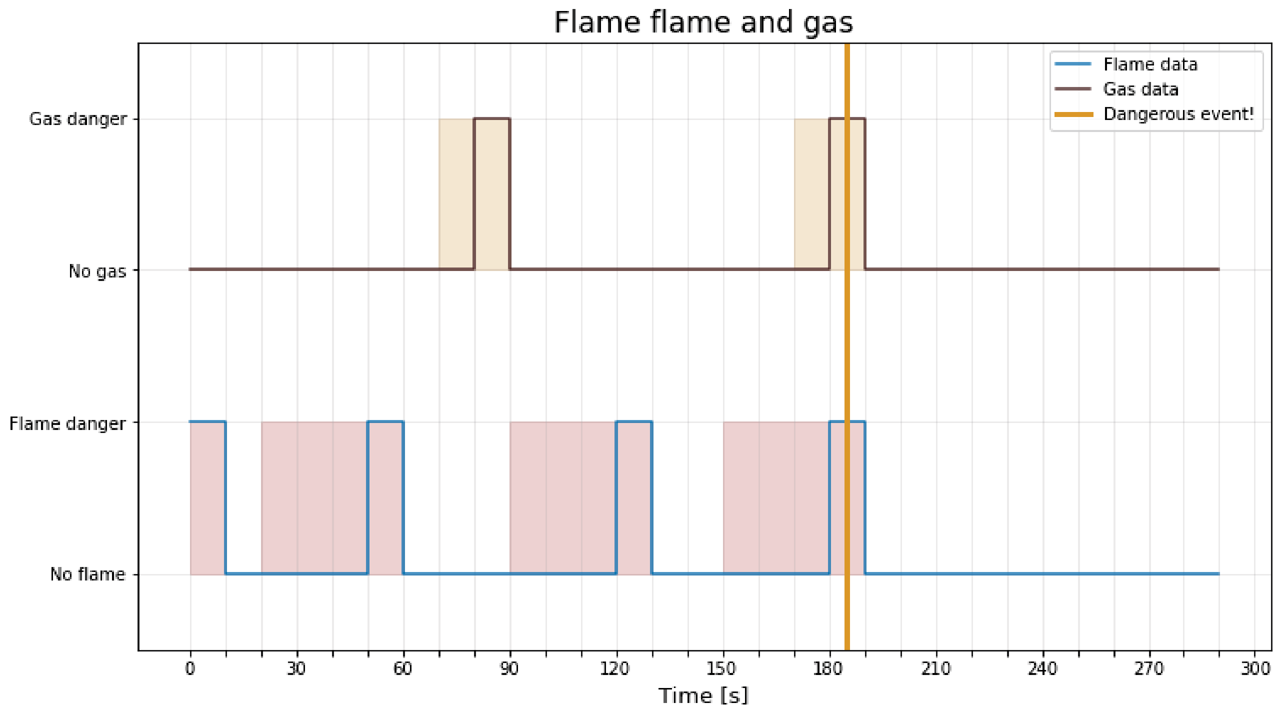


Figure 6.8: Figure shows another simulation of gas and flame sensors data. In this case there is a simultaneous detection and the algorithm prompts a notification about it.

Chapter 7

Applications and possible improvements

As for now, possible uses of the solution can include jobs involving working in an environment with presence of flammable gas, dangerous heights, moving machines, restricted areas. Furthermore, the described solution has to be organized and implemented in a fixed environment (e.g. a factory), in which work space, rooms, people roles and accesses are studied and integrated with the software (e.g. to control accesses to different areas or to lead people to nearest exit in case of danger). Moreover, a control room should be present with personnel that can control and help out with emergency resolutions. For these reasons, the application of this kind of solution is most useful when working in large environments with many people and where dangers can spread very easily (e.g. a gas plant or a factory handling flammable gas); in these cases the studied solution can spot dangers, notify all present users about them and it can manage the proper actions for everyone, including guiding people towards nearest exit in a ordered way, helping firefighter users to handle the beginning of fires, leading first aid people to assist injured users.

In other different cases the application of the wearable solution can have less positive impact; e.g. in case of a small working environment which usually present a small number of people, chaos handling is less important since the risk of causing chaos during crisis is lower; emergency exits are easier to find and danger situation in general are easier to handle; in case of slow-spreading dangers, traditional safety systems can be used and workers can be advised via the standard alert methods.

As mentioned before, another unfavorable point is that the system should be integrated in the proper scenario where people act; going outside the predefined zones implies that the system is not able to act properly. The connection can be lost and so data are not received nor sent to user; notifications becomes useless since the outside places are not registered to the software (e.g. those involving emergency exits and access control). A way the other features of the wearable (detection of *man-down* events, presence of flames or flammable gasses) can be exploited everywhere involves the use of SIM cards in Raspberry module and Smartwatch, thus enabling them to send and receive data from all areas covered by an internet connection.

Looking for future improvements, presented solution can be updated to be applicable in more different scenarios.

For example, in case of heavy smoke it becomes very hard to see also the nearest objects; in a scenario where a firefighter team is called for a fire event involving people trapped or lost in a building, if they are able to enter and look for them, rescue procedure may be difficult due to scarce vision. A possible solution can be to integrate an ultrasonic sensor which acts like a radar in the wearable. It can be both integrated with the Raspberry or stand-alone. The crucial point is it sends information to a server (it can also be a mobile center that can be installed on the firefighter truck to be always available) which then transmits it to the smart-glasses of the team (supposing they wear them). This way they can move in the building despite heavy smoke situation and are able to better help the people in trouble.

Another possible improvement involves the usage of a camera and/or an infra-red (IR) camera.

They can be integrated with the Raspberry module and they can be enabled if necessary; for example, in a *man down* case where there are no room cameras near the intervention scenario, a camera on the wearable can be enabled to send visual information about the trauma and the positioning of the injured user. The IR camera instead can be exploited (with the help of smart glasses or a visor) in case the lights go out or if users have to move in a dark surroundings. It can also be useful in particular situations such as finding out possible gas leaks using visual images.

Other possible improvements involving the augmented reality related to the smart glasses are the possibility to save every-day backup of the rooms shapes and objects positioning; these can be used when, for example, some thieves enter in the workplace and steal some valuable items. Helped by the backup and augmented reality, an employee could quickly notice what is missing. Another example of application, when doable, is to use this backups in case of a disaster and subsequent building collapse; that backup, along with GPS information and details about people roles and positioning, can help rescuing trapped users much faster.

Chapter 8

Conclusions

This work presented a possible implementation of a wearable solution for personal safety. *Use cases* in which it would be useful were studied, highlighting how this work can help in handling special situations, e.g. access control and possible injuries detected by *integration-platform* algorithm, combined with the messages coming from the third-party platform.

Hardware and sensors were selected basing on the possible applications of the solution trying to include the detection for most common and dangerous risks, such as the presence of gas leaks, that are usually hardly perceivable by human beings. A scope of this work was to study the Raspberry module and its way to collect data from sensors. The interaction between hardware and software was analyzed; sensors were successfully tested to collect data. Thanks to this, a program collecting information from multiple sensors was written, enabling the Raspberry module to gather different environmental data simultaneously. Moreover, it can be modified and expanded to collect data from more sensors, allowing for future improvements.

Following IoT paradigm, Raspberry module was tested to send data to the integration-platform, thus enabling data sharing in the network; in particular, the integration-platform is meant to send data and notifications to a dashboard; it is made to show recent data in a control room. Here people can see the alerts and handle emergency situation faster thanks to the data they have. The integration platform can also be enabled to send notifications to specific devices connected to the network (e.g. users smartphones and smart objects), thus alerting in a more precise way than usual alarm systems. This way, notified users know the situation they are facing and can handle it with less panic.

Unfortunately, data is scarce due to the impossibility to simulate certain events. Future developments can concern data collection and storage; data analysis can be carried out try highlight specific patterns in some parameters, as it is showed in this work concerning heart rate data and sound recordings.

Platform algorithm was tested simulating data coming from sensors. The results showed it works as expected, upholding its potential of combining information coming from different sources. Nevertheless, for the future, real situation data acquisitions are needed to confirm the correct functioning of the platform in real scenarios and to better tune the parameters triggering the notifications.

The presented device can be used in many different kinds of scenarios involving dangers spot and some emergency situation. Its applications can improve the safety conditions during everyday job routine of many workers which face daily dangerous situations. This thesis presented only a first development of this solution. It wanted to show that such a device is actually doable and that it can be integrated successfully with a back-end system, leaving the door opened for more studies and possible developments which can also differ from the initial purposes. Some examples can be adding the possibility to include a backup of the whole system (data, platforms, algorithms, ecc.), studying how to decrease the possibilities of a system crash and eventually how to notify such a failure. Further studies can also be carried out adding new sensors at the wearable, thus increasing the data gathering mechanism and allowing for more parameters to be analyzed (e.g. humidity percentage, luminosity). Furthermore, thanks to the choice of Raspberry device, the solution can be enriched using some camera modules (standard camera and IR) and adapted to collect data from them. This can improve handling of already presented scenarios (e.g. sending images to the control room concerning how a injured user is positioned and the king of wound he has) and introducing the possibility of new applications (such as being able to see in the dark, integrating the IR camera with the smart glasses, or spotting gas leaks exploiting IR images).

Appendix A: installing libraries

In this Appendix it is briefly shown how to download, install and verify the installation of the *wiringPi* and *GPIO* libraries. They are needed to make C and/or Python scripts able to interact with sensors in a proper way. [21]

wiringPi

WiringPI is a C language GPIO library applied to the Raspberry Pi platform. It complies with GNU Lv3. If the latest Raspbian is installed, library installation can be skipped.

Download To download the library, in an open terminal the following instruction must be typed:

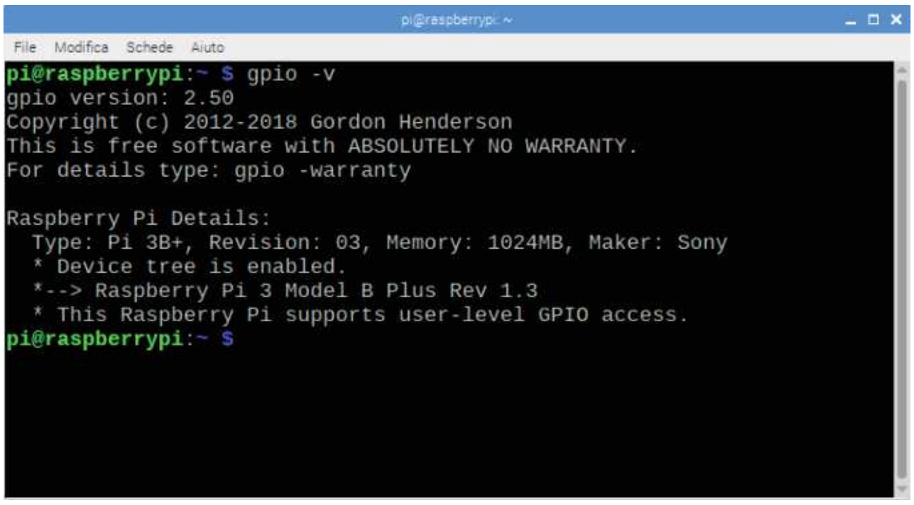
```
git clone git://git.drogon.net/wiringPi
```

Install Immediately after the previous step, typing the following lines will install the library:

```
cd wiringPi
git pull origin
./build
```

Testing installation The correct installation can be verified by typing:

```
gpio -v
```

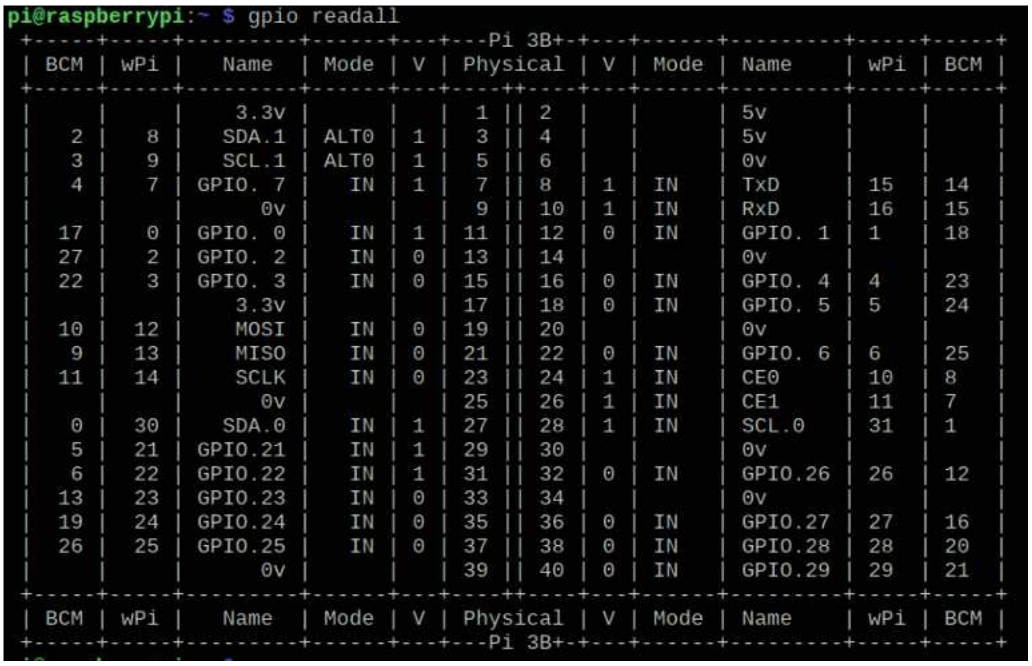


```
pi@raspberrypi:~ $ gpio -v
gpio version: 2.50
Copyright (c) 2012-2018 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
  Type: Pi 3B+, Revision: 03, Memory: 1024MB, Maker: Sony
  * Device tree is enabled.
  *--> Raspberry Pi 3 Model B Plus Rev 1.3
  * This Raspberry Pi supports user-level GPIO access.
pi@raspberrypi:~ $
```

If the message displayed above appears, the wiringPi is installed successfully.

To visualize all the different name convention referring to the pins, the following command has to be typed.

```
gpio readall
```



```
pi@raspberrypi:~ $ gpio readall
 +-----+-----+---------+------+---+---Pi 3B+-+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5v      |     |     |
 |   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | IN   | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | IN   | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 1 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+---Pi 3B+-+---+------+---------+-----+-----+
```

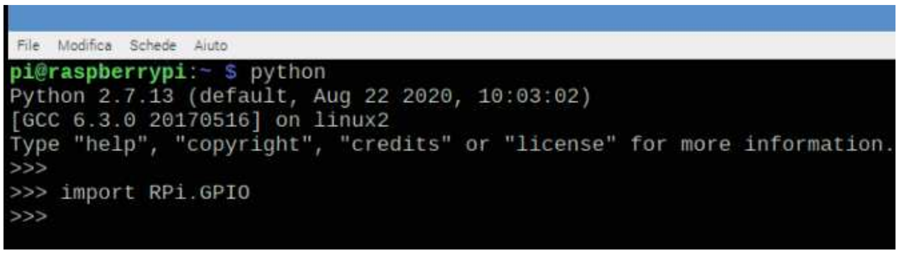
RPi.GPIO

For Python users, GPIOs can be programmed with API provided by RPi.GPIO. It is a module to control RaspberryPi GPIO channels. The package provides a class to control the GPIO on a RaspberryPi. For examples and documents, refer to [22] (pag 39 of [20]).

Installing In case the library is not installed on the system, the following commands can be typed in a terminal [23]:

```
sudo apt-get update
sudo apt-get install python-rpi.gpio python3-rpi.gpio
```

Testing installation To verify the correct installation it can be imported in python:



```
pi@raspberrypi:~ $ python
Python 2.7.13 (default, Aug 22 2020, 10:03:02)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> import RPi.GPIO
>>>
```

If no error is displayed, than the library is installed successfully (pag 39 of [20]).

Appendix B: code for data collection

The python script used for data collection is reported here.

```
#!/usr/bin/env python3.7
import PCF8591 as ADC
import RPi.GPIO as GPIO
import time
import math


# Input from flame sensor
DF = 4
# Input from gas sensor
DO = 17
GPIO.setmode(GPIO.BCM)

def setup():
# Setup for AD converter
ADC.setup(0x48)
# Setup for FLAME GPIO pins
GPIO.setup(DF, GPIO.IN)
# Setup for GAS GPIO pins
GPIO.setup(DO, GPIO.IN)

def Print_flame(x):
   if x == 1:
      print('  * Fire Safe *')
   if x == 0:
      print('  * Fire! *')


def Print_temp(x):
   if x == 1:
      print('* Better~ *')
   if x == 0:
      print('* Too Hot! *')

def Print_gas(x):
   if x == 1:
      print('  * Gas Safe *')
   if x == 0:
      print('  * Danger Gas! *')


def loop():

   status_flame = 1
   tmp_flame = 1

   status_temp = 1
   tmp_temp = 1
```

```
status_gas = 1
count_gas = 0

while True:


  # N.B.: reading FLAME from AIN0
  #  reading TEMP from AIN1
  #  reading SOUND from AIN2
  #  reading GAS from AIN3


  # FLAME DETECTION
  print("Analog flame:", ADC.read(0))

  tmp_flame = GPIO.input(DF);
  if tmp_flame != status_flame:
  Print_flame(tmp_flame)
  status_flame = tmp_flame


  # TEMPERATURE DETECTION
  analogVal = ADC.read(1)
  print("Analog temp:", analogVal)

  Vr = 5 * float(analogVal) / 255
  Rt = 10000 * Vr / (5 - Vr)
  temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
  temp = temp - 273.15
  print('temperature = ', temp, 'C')

  # For Thermister module(with sig pin)
  if temp > 33:
  tmp_temp = 0
  elif temp < 31:
  tmp_temp = 1
  ##################################################

  if tmp_temp != status_temp:
  Print_temp(tmp_temp)
  status_temp = tmp_temp


  # SOUND DETECTION
  count = 0
  voiceValue = ADC.read(2)
  if voiceValue:
  print ("Noise value:", voiceValue)
  if voiceValue < 50:
  print ("Voice detected! ", count)
  count += 1

  # GAS DETECTION
  gasVal = ADC.read(3)
  print("Gas value:", gasVal)

  tmp_gas = GPIO.input(DO)
  print(tmp_gas)
  if tmp_gas != status_gas:
  Print_gas(tmp_gas)
```

```
        status_gas = tmp_gas

        # STORING VALUES TO FILE
        ifile = open("/home/pi/test.txt","w")

        ifile.write("%d3.2;%3i;%3i" % (temp, voiceValue, gasVal))

        ifile.close()

        time.sleep(2)

if __name__ == '__main__':
   try:
      setup()
      loop()
   except KeyboardInterrupt:
      pass
```

The main idea of this script is to read the sensors nearly every 2 seconds and write the values to a file. The same file will be read by a *Node-RED* flow. As already explained, frequency modulation will be studied better when the application will be tested in real situations, hopefully including different devices to test the network stability.

Appendix C

Here the results of preliminary tests are reported. Figures show screen output generated by the code while detecting information from the sensors. Temporal development is from top to bottom, last measures are those in the bottom lines.

Flame sensor

For the flame sensor, a lighter was used to test its sensitivity. It was ignited and moved towards the sensor. As shown in the figure below, the signal changed as the lighter went closer. The measures, as the above code shows, are detected every 0.5 seconds (Figure 8.1).

Gas sensor

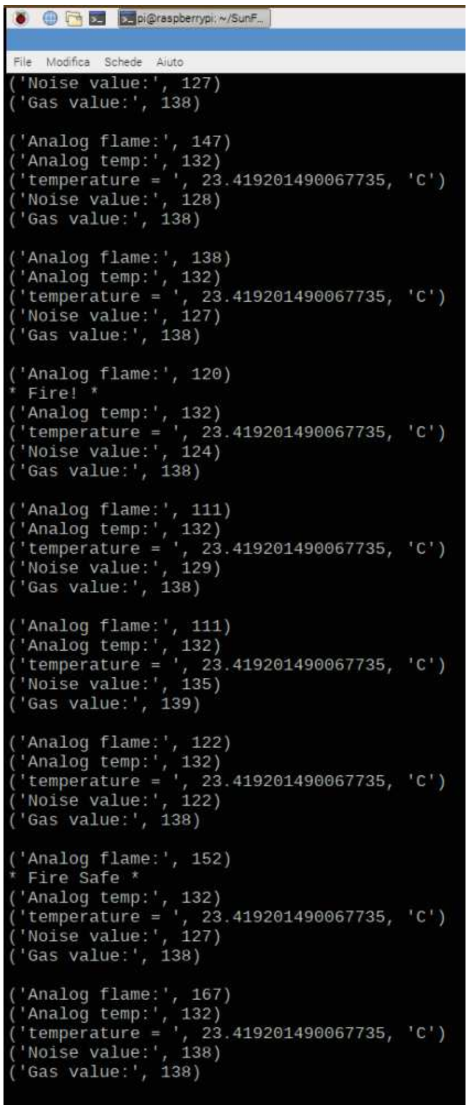
To test gas sensor functioning, lighter gas was used. A lighter was put near the detector and gas was released (without igniting it). Sensor response was detected by the script (Figure 8.2). The sensor needs to be heated up for a while before taking measures; for this reason, it was left turned on for some minutes until the values on the screen becomes stable in time.

Sound sensor

To test sound sensor functioning, some noise was made near it using voice and hitting the surface it was laying on. Something that can be noticed is that the response of this sensor is different from the others; it measures the instant vibration it perceives, making the record very hard to achieve since if there is no instantaneous external solicitation, the membrane lays at rest and no noise is detected (Figure 8.3a).

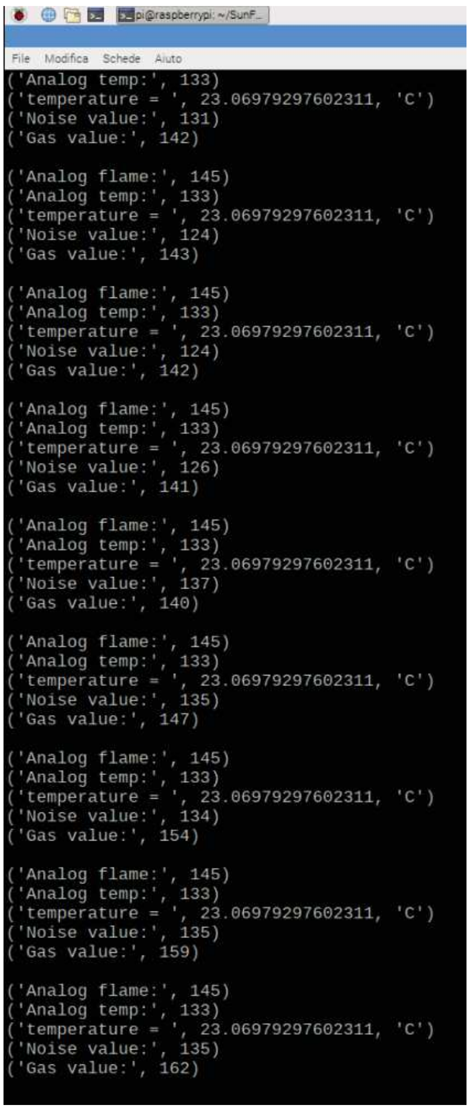
Temperature sensor

To test the temperature sensor functionality, it was simply left exposed to environment, then touched, and left again. When in contact with the hand, temperature started raising (Figure 8.3b). Temperature values printed to screen seemed realistic so it was assumed the conversion algorithm was written properly by the authors (conversion lines were taken from an example script [26]).



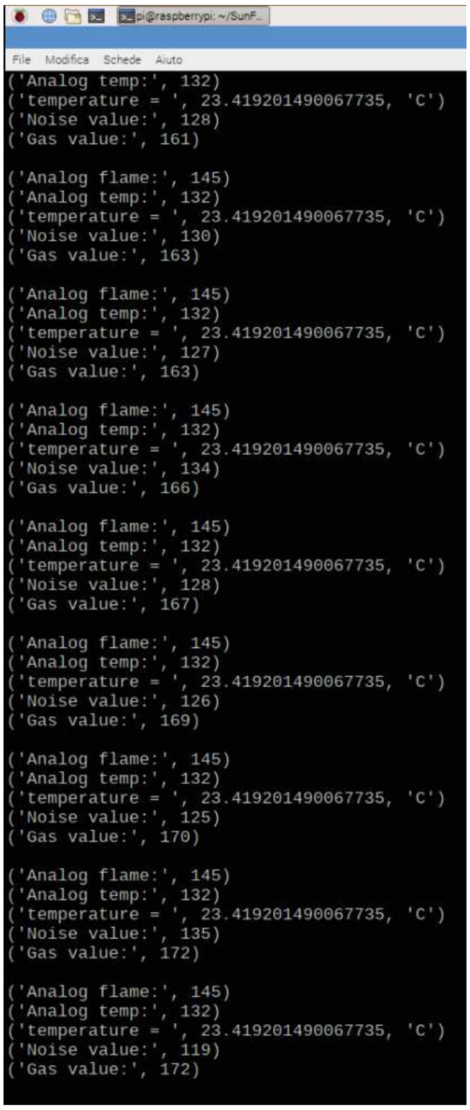
```
('Noise value:', 127)
('Gas value:', 138)

('Analog flame:', 147)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 128)
('Gas value:', 138)

('Analog flame:', 138)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 127)
('Gas value:', 138)

('Analog flame:', 120)
* Fire! *
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 124)
('Gas value:', 138)

('Analog flame:', 111)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 129)
('Gas value:', 138)

('Analog flame:', 111)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 135)
('Gas value:', 139)

('Analog flame:', 122)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 122)
('Gas value:', 138)

('Analog flame:', 152)
* Fire Safe *
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 127)
('Gas value:', 138)

('Analog flame:', 167)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 138)
('Gas value:', 138)
```

Figure 8.1: Figure shows script output when a flame is ignited (in this case a lighter is used) near flame sensor. The sensor value changes properly and when it overcomes the threshold "Fire!" state is prompted to screen. When the signals detected by the sensor are no longer above the threshold, a "Fire Safe" state is prompted, signaling that the dangerous situation is no longer present.



```
('Analog temp:', 133)
('temperature = ', 23.06979297602311, 'C')
('Noise value:', 131)
('Gas value:', 142)

('Analog flame:', 145)
('Analog temp:', 133)
('temperature = ', 23.06979297602311, 'C')
('Noise value:', 124)
('Gas value:', 143)

('Analog flame:', 145)
('Analog temp:', 133)
('temperature = ', 23.06979297602311, 'C')
('Noise value:', 124)
('Gas value:', 142)

('Analog flame:', 145)
('Analog temp:', 133)
('temperature = ', 23.06979297602311, 'C')
('Noise value:', 126)
('Gas value:', 141)

('Analog flame:', 145)
('Analog temp:', 133)
('temperature = ', 23.06979297602311, 'C')
('Noise value:', 137)
('Gas value:', 140)

('Analog flame:', 145)
('Analog temp:', 133)
('temperature = ', 23.06979297602311, 'C')
('Noise value:', 135)
('Gas value:', 147)

('Analog flame:', 145)
('Analog temp:', 133)
('temperature = ', 23.06979297602311, 'C')
('Noise value:', 134)
('Gas value:', 154)

('Analog flame:', 145)
('Analog temp:', 133)
('temperature = ', 23.06979297602311, 'C')
('Noise value:', 135)
('Gas value:', 159)

('Analog flame:', 145)
('Analog temp:', 133)
('temperature = ', 23.06979297602311, 'C')
('Noise value:', 135)
('Gas value:', 162)
```

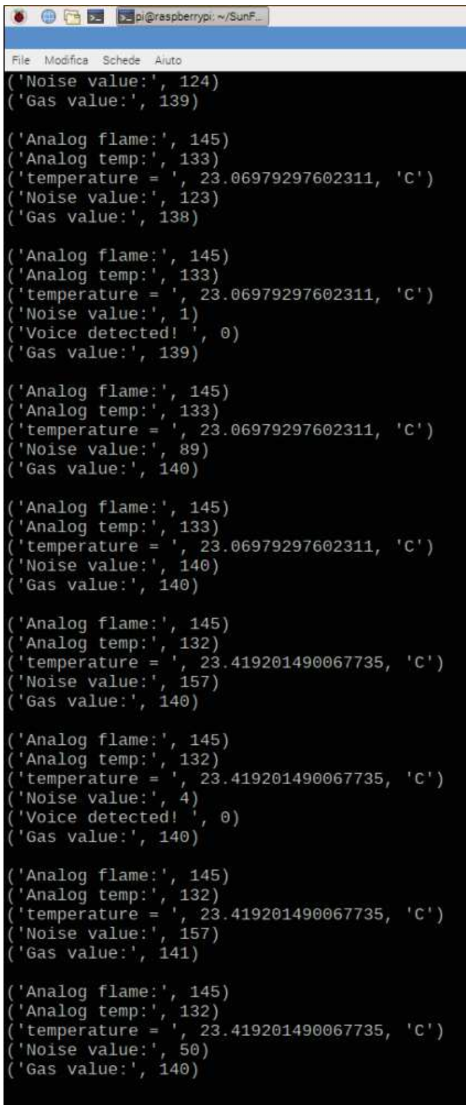
(a)



```
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 128)
('Gas value:', 161)

('Analog flame:', 145)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 130)
('Gas value:', 163)

('Analog flame:', 145)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 127)
('Gas value:', 163)

('Analog flame:', 145)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 134)
('Gas value:', 166)

('Analog flame:', 145)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 128)
('Gas value:', 167)

('Analog flame:', 145)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 126)
('Gas value:', 169)

('Analog flame:', 145)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 125)
('Gas value:', 170)

('Analog flame:', 145)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 135)
('Gas value:', 172)

('Analog flame:', 145)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 119)
('Gas value:', 172)
```

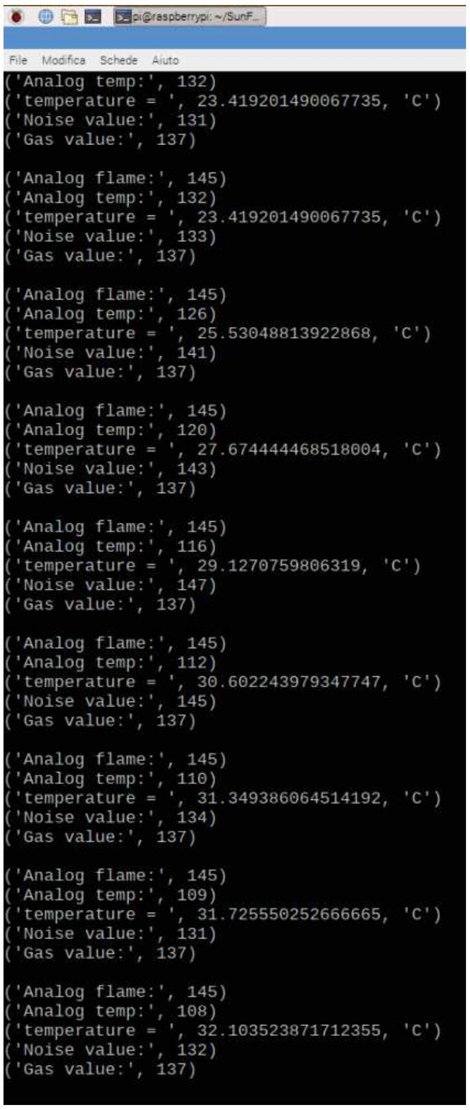
(b)

Figure 8.2: Figures show script output while testing gas sensor. In this case gas was released near the detector surface; the changes in sensor values can be noticed in both (a) and (b) (from top to bottom). If the gas concentration reaches a certain threshold, "Danger Gas!" will be printed on the screen. In our case, we can notice how the value registered by the sensor is increasing with time, but the threshold is not reached. This limit can be modified using the potentiometer on MQ-2 gas sensor module.



```
('Noise value:', 124)
('Gas value:', 139)

('Analog flame:', 145)
('Analog temp:', 133)
('temperature = ', 23.06979297602311, 'C')
('Noise value:', 123)
('Gas value:', 138)

('Analog flame:', 145)
('Analog temp:', 133)
('temperature = ', 23.06979297602311, 'C')
('Noise value:', 1)
('Voice detected! ', 0)
('Gas value:', 139)

('Analog flame:', 145)
('Analog temp:', 133)
('temperature = ', 23.06979297602311, 'C')
('Noise value:', 89)
('Gas value:', 140)

('Analog flame:', 145)
('Analog temp:', 133)
('temperature = ', 23.06979297602311, 'C')
('Noise value:', 140)
('Gas value:', 140)

('Analog flame:', 145)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 157)
('Gas value:', 140)

('Analog flame:', 145)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 4)
('Voice detected! ', 0)
('Gas value:', 140)

('Analog flame:', 145)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 157)
('Gas value:', 141)

('Analog flame:', 145)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 50)
('Gas value:', 140)
```

(a)



```
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 131)
('Gas value:', 137)

('Analog flame:', 145)
('Analog temp:', 132)
('temperature = ', 23.419201490067735, 'C')
('Noise value:', 133)
('Gas value:', 137)

('Analog flame:', 145)
('Analog temp:', 126)
('temperature = ', 25.53048813922868, 'C')
('Noise value:', 141)
('Gas value:', 137)

('Analog flame:', 145)
('Analog temp:', 120)
('temperature = ', 27.674444468518004, 'C')
('Noise value:', 143)
('Gas value:', 137)

('Analog flame:', 145)
('Analog temp:', 116)
('temperature = ', 29.1270759806319, 'C')
('Noise value:', 147)
('Gas value:', 137)

('Analog flame:', 145)
('Analog temp:', 112)
('temperature = ', 30.602243979347747, 'C')
('Noise value:', 145)
('Gas value:', 137)

('Analog flame:', 145)
('Analog temp:', 110)
('temperature = ', 31.349386064514192, 'C')
('Noise value:', 134)
('Gas value:', 137)

('Analog flame:', 145)
('Analog temp:', 109)
('temperature = ', 31.725550252666665, 'C')
('Noise value:', 131)
('Gas value:', 137)

('Analog flame:', 145)
('Analog temp:', 108)
('temperature = ', 32.103523871712355, 'C')
('Noise value:', 132)
('Gas value:', 137)
```

(b)

Figure 8.3: Figures show sensor records while testing sound sensor (a) and thermistor functioning (b). As one can notice in figure (a), sound detection module acts in different way with respect to the others. This is due to its functioning principle, which makes it detect only almost instantaneous signals. Looking at the script output indeed, it can be noticed how sometimes the signals are similar with respect to sensor *baseline* even if some noise is being made; if the detected vibration causes a sufficient potential variation, the signal is interpreted as noise and "Voice detected!" is printed to screen.5 Figure (b) shows the raising temperature while the sensor has just been touched (and it is still in contact with the heat source, a hand in this case). The temperature printed seems reasonable. As in previous cases, if a certain prefixed threshold is exceeded, a signal will be written to screen.

Appendix D

Here the code of raspberry flow main node is reported. At beginning, the message from previous node is read and values are stored in a list, split by semicolumn ";", as it was chosen to save data in this way. Current date and time are saved to be inserted in the message. Some variables are created to make test on read values and apply some changes to the payload accordingly. Main body of the message is prepared and, after that, tests are done to verify the presence of dangers. If there are some, payload values are changed consequently; furthermore, some notifications are added alerting about the exceeded thresholds.

```
var list_param = String(msg.payload);
list_param = list_param.split(";");

// Order of parameters:
//  flame
//  gas
//  noise
//  temp

let dateTime = Date.now();

let flame_value = false;
let gas_value = false;
let sound_value = false;

// Preparing msg structure
msg = {
   payload:{
      DEVICE_ID: "RaspberryPi_field",
      SOURCE: "fieldSensors",
      POSITION: {
         LATITUDE: "",
         LONGITUDE: ""
      },
      TIMESTAMP: dateTime,
      SIGNALS: [{
         TYPE: "flame",
         VALUE: flame_value,
         MIN_THRESHOLD: null,
         MAX_THRESHOLD: null,
         UNIT_OF_MEASUREMENT: null,
         SIGNAL_TIMESTAMP: dateTime,
      },
      {
         TYPE: "gas",
         VALUE: gas_value,
         MIN_THRESHOLD: 0,
         MAX_THRESHOLD: null,
         UNIT_OF_MEASUREMENT: "K",
         SIGNAL_TIMESTAMP: dateTime,
      },
      {
         TYPE: "sound",
         VALUE: sound_value,
         MIN_THRESHOLD: 0,
         MAX_THRESHOLD: null,
         UNIT_OF_MEASUREMENT: "K",
         SIGNAL_TIMESTAMP: dateTime,
      },
      {
         TYPE: "temperature",
         VALUE: list_param[3],
         MIN_THRESHOLD: 0,
         MAX_THRESHOLD: null,
         UNIT_OF_MEASUREMENT: "K",
         SIGNAL_TIMESTAMP: dateTime,
      }],
      NOTIFICATIONS: []
   }
}


//Flame sensor
// N.B.: safe signal is 1
if (list_param[0] === '0') {
   flame_value = true;
   msg.payload.SIGNALS[0].VALUE = true;
   msg.payload.NOTIFICATIONS.push({
      TYPE: "event",
      VALUE: "detected fire",
      PRIORITY: 1,
      ROLE: "operator",
      TEXT_COLOR: "",
      BACKGROUND_COLOR: "",
      VIBRATE: ""
   }
   );
}

//Gas sensor
if (list_param[1] === '0') {
   gas_value = true;
   msg.payload.SIGNALS[1].VALUE = gas_value;
   msg.payload.NOTIFICATIONS.push({
      TYPE: "event",
      VALUE: "detected falmmable gas",
      PRIORITY: 1,
      ROLE: "operator",
      TEXT_COLOR: "",
      BACKGROUND_COLOR: "",
      VIBRATE: ""
   }
   );
}

//Sound sensor
if (list_param[2] === '0') {
   sound_value = true;
   msg.payload.SIGNALS[2].VALUE = sound_value;
   msg.payload.NOTIFICATIONS.push({
      TYPE: "event",
      VALUE: "detected noise",
        PRIORITY: 1,
        ROLE: "operator",
        TEXT_COLOR: "",
        BACKGROUND_COLOR: "",
        VIBRATE: ""
    }
    );

}


//Temperature sensor
if (list_param[3] >= 35) {
    msg.payload.NOTIFICATIONS.push({
        TYPE: "event",
        VALUE: "detected high temperature",
        PRIORITY: 1,
        ROLE: "operator",
        TEXT_COLOR: "",
        BACKGROUND_COLOR: "",
        VIBRATE: ""
    }
    );

}

return msg;
```

Bibliography

[1] https://www.pwc.com/gx/en/about.html (last visit: 06/09/2020)

[2] https://eicsummit.com/ (last visit: 06/09/2020)

[3] https://www.spsitalia.it/it/la-fiera (last visit: 06/09/2020)

[4] www.sunfounder.com (last visit: 16/09/2020)

[5] https://www.samsung.com/it/wearables/galaxy-watch-r800/ (last visit: 16/09/2020)

[6] https://www.sunfounder.com/learn/sensor-kit-v2-0-for-raspberry-pi-b-plus/lesson-18-temperature-sensor-sensor-kit-v2-0-for-b-plus.html (last visit: 17/09/2020)

[7] http://wiki.sunfounder.cc/index.php?title=Thermistor_Module (last visit: 17/09/2020)

[8] http://wiki.sunfounder.cc/images/4/49/Thermistor_datasheet.pdf (last visit: 17/09/2020)

[9] https://www.sunfounder.com/learn/sensor-kit-v2-0-for-raspberry-pi-b-plus/lesson-19-sound-sensor-sensor-kit-v2-0-for-b-plus.html (last visit: 17/09/2020)

[10] http://wiki.sunfounder.cc/index.php?title=Sound_Sensor_Module (last visit: 17/09/2020)

[11] https://www.sunfounder.com/learn/sensor-kit-v2-0-for-raspberry-pi-b-plus/lesson-22-gas-sensor-sensor-kit-v2-0-for-b-plus.html (last visit: 17/09/2020)

[12] https://components101.com/mq2-gas-sensor (last visit: 11/10/2020)

[13] http://wiki.sunfounder.cc/index.php?title=MQ-2_Gas_Sensor_Module (last visit: 11/10/2020)

[14] https://www.sunfounder.com/learn/sensor-kit-v2-0-for-raspberry-pi-b-plus/lesson-21-flame-sensor-sensor-kit-v2-0-for-b-plus.html (last visited: 11/10/2020).

[15] http://wiki.sunfounder.cc/index.php?title=Flame_Sensor_Module (last visited: 11/10/2020).

[16] https://www.samsung.com/us/heartratesensor/ (last visit: 20/09/2020)

[17] http://wiki.sunfounder.cc/index.php?title=PCF8591_8-bit_A/D_and_D/A_converter_Module (last visit 21/09/2020).

[18] https://www.samsung.com/it/wearables/galaxy-watch-r800/

[19] https://www.cdc.gov/nchs/data/nhsr/nhsr041.pdf

[20] https://www.sunfounder.com/media/download_file/English-Sensor_kit_V2.0_for_Raspberry_Pi_4_Model_B_2020.07.08.pdf (last visit 22/09/2020).

[21] http://wiringpi.com/download-and-install/ (last visit 22/09/2020).

[22] http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/ (last visit 22/09/2020).

[23] https://sourceforge.net/p/raspberry-gpio-python/wiki/install/ (last visit 22/09/2020).

[24] https://www.sunfounder.com/learn/category/sensor-kit-v2-0-for-raspberry-pi-b-plus.html (last visit 23/09/2020).

[25] https://nodered.org/ (las visit 23/09/2020).

[26] https://www.sunfounder.com/learn/sensor-kit-v2-0-for-raspberry-pi-b-plus/lesson-18-temperature-sensor-sensor-kit-v2-0-for-b-plus.html (last visit 06/10/2020).

[27] http://www.laboratoriopoliziademocratica.org/SALUTE/GIMLE_rischio_rumore_1.7.09.pdf