Activity and Fall Detection in the Habitational Environment

Subsystem: Interface

S. Delfos and E. Granneman

Activity and Fall Detection in the Habitational Environment

Subsystem: Interface

by

S. Delfos and E. Granneman

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended on July 1, 2019.

Student number:	4317262 4227484	
Project duration:	May 23, 2019 – July 5, 2019	
Thesis committee:	ir. E.W. Bol,	TU Delft, Chair
	Prof. dr. P.J. French,	TU Delft, Supervisor, Proposer
	Dr. M. Ghaffarian Niasar	TU Delft, Jury member
	ir. K. Rassels,	TU Delft, Jury member

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.
The source code can be received on request

TU Delft Delft University of Technology

Abstract

This report describes the design and implementation of an interface subsystem for a fall detection system using a pressure based floor sensor. The goal of the fall detection system is to detect and alarm when an elderly person has fallen.
The subsystem covers communication between hardware and software, using the WiFi protocol MQTT. The communication is fast enough so that it does not limit other subsystem performance and allows for high expandability to accommodate the hardware's modular design. The interface module receives a probability of fall occurrence from the separate processing module, from which an average is taken over a 30 second time window to avoid false positives. Alarming is done through the use of text-to-speech voice API to make automated phone calls that retrieve user input to confirm or deny that help is on the way. A habit tracker proof of concept is provided that could improve the accuracy of fall detection by checking if the client is showing anomalies during a given day. A user interface is implemented using the Kivy environment. This environment has its own language which allows for separation of functionality and layout, which keeps the interface elements separate and leads to ease of use.
Overall, this results in a strong alternative for fall detection that could be used to improve the time an elderly person can live at home safely without the need to move to a nursing home.

Preface

We would like to thank our project proposer and supervisor prof. dr. Paddy J. French for the help and freedom within the project. We are very grateful for ir. Kianoush and dr. ir. Gerard J. M. Janssen because of their valuable feedback and support during the project.

Many thanks go out to the Python community for providing documentation for many packages used in the project.

Also, we would like to thanks our fellow project members I. Cornelis, S. Falkena, B. den Oude and H.J. Kruisse for their excellent work within the project, healthy discussions and overall great teamwork.

S. Delfos and E. Granneman
Delft, June 2019

Contents

1 Introduction

Falls of adults above the age of 65 are the leading cause of head injuries and broken hips, with one out of ten falls resulting in serious injuries [1]. This comes at a large medical care cost for society. Furthermore, these falls often go unnoticed for longer than necessary, and sending help earlier can prevent a large number of serious injuries [2]. Of course, preventing falling in the first place would be ideal. However, a single solution is impossible since falling has many causes. However, the result in many cases is the same: a person lies on the floor. Therefore, a more general solution that detects the effect, instead of the movement of falling, is easier to implement and can also help with reducing medical costs.

1.1. Project objective

To implement an activity [1] and fall detection system, an Electrical Engineering Bachelor End Project was proposed. This project is executed by six students, who need to design, implement and test a fall detection system. The project is split up in 3 subgroups of 2: The first subgroup has been responsible for the hardware design [3], the second group has been responsible for the interfacing, alarming and activity tracking [4], while the last group has been responsible for developing the fall detection algorithm [5].

1.2. Thesis outline

This chapter will describe the background information and current state of the art technology. After this, the global system requirements are presented in chapter 2, followed by the initial design decision in chapter 3. Chapter 4 introduces the Interface subsystem and presents the subsystem requirements, ensuing from this the five separate modules hardware connection, pre-processing, habit tracker, alarming and user interface are described in chapter 5, 6, 7, 8 and 9. Finally, the subsystem is evaluated and discussed followed by future work and recommendations in chapter 10. A global system evaluation is given in chapter 11.

1.3. Background information

In the Netherlands in 2017, 3849 deaths among the elderly (65+) were because of falling [6]. Worldwide, falls are the second leading cause of accidental or unintentional injury deaths [7]. Therefore, fall prevention has been researched by many organizations, such as the WHO (World Health Organization) [8]. As stated before, not all fall incidents can be prevented. When an inevitably fall happens, it is undesirable that someone remains on the ground for longer periods. For example, when a bone fracture is caused by a fall, the person should not try to stand up by themselves. Moving around with broken bones can increase pain and bleeding and can damage tissues around the injury. This can lead to complications in the repair and healing of the injury later on [9]. Furthermore, elderly could be in shock.

In research by Fleming et al. [10], 54% (144/265) of falls reported described the participant as being found on the floor and 82% (217/265) of falls occurred when the person was alone. It was found that of the people who fell, 80% were unable to get up after at least one fall and 30% had lain on the floor for an hour or more. It can be seen from table 1.1 that 83% of the participants were alone and unable to get up after 5 minutes. This research shows that a fall detection system can be useful.

[1] Activity is referred to as the indoor translocation of a person

Table 1.1: Time on the floor after fall during one-year follow-up. Figures are percentages of falls

Time on floor	All Falls (n = 265)	Participant Alone (n = 217)	Participant unable to get up (n=176)	Participant alone and unable to get up (n=143)
<5 min.	43%	36%	26%	17%
5 min. - 1 hr.	36%	39%	44%	48%
1-2 hr.	5%	6%	8%	10%
>2 hr.	10%	12%	15%	18%
Unknown	6%	7%	7%	7%

The fear of falling can lead to people deciding to move to an elderly care home. This has a great financial impact on the Dutch government as can be seen from figure 1.1 [11]. From this figure, it can be concluded that it is roughly 3 times less expensive for the government to keep people living at home (scale 4) instead of moving to an elderly home (scale 5).

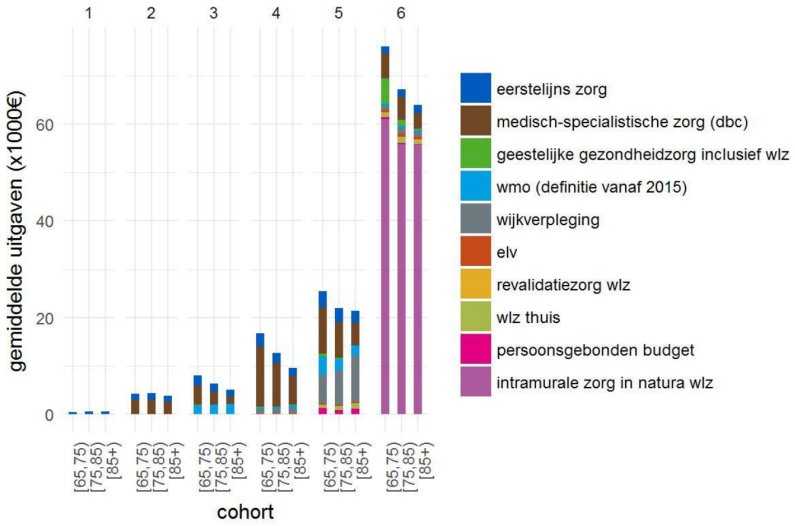

Figure 1.1: Care costs for the Dutch government in 2018

1.4. Definition of a fall

For a system that has to decide whether a person has fallen, it is very important to define properly what a fall is. First, a distinction needs to be made between falling and a fall. Falling is the act of coming to the ground, while a fall is the result of falling. Two papers define falling as:

> *"The rapid change from the upright or sitting position to the reclining or almost lengthened position, but not a controlled movement, like lying down."* [12]

In 1987 Gibson additionally defined falling as:

> *"Unintentionally coming to the ground, or some lower level not as a consequence of sustaining a violent blow, loss of consciousness, sudden onset of paralysis as in stroke or an epileptic seizure."* [13]

However, this definition should be extended to include falls resulting from dizziness and syncope. Thus, a fall is not necessarily the result of a sudden change of body position, but could also be due to a slow collapse. Therefore, trying to detect a fall instead of falling is the more overarching solution.

1.5. Current state of the art solutions

Currently, fall detection is mostly done using wearable methods and cameras [14]. However, research has been done into other directions such as Floor mounted technology, Radar technology, more advanced vision technology, and Seismic technology.

1.5.1. Wearables

Most available wearable solutions use an accelerometer to detect a fall, such as the Philips Lifeline series [15]. Similar to the Philips solution, Yacchirema et al. [16] used an accelerometer and combined this with machine learning. The software was trained using publicly available data of a triaxial accelerometer in various scenarios [17]. Another way to detect falls using a wearable solution is to measure vertical velocity, again using an accelerometer. This was successfully demonstrated by Lee et al. [18].

Advantages of these systems can be found in the ease of detection, no requirement to alter the home and the ability for the person wearing them to call for help whenever they feel they require it. Commonly, wearable technology is cheaper compared to other solutions as well. Disadvantages of these systems are that they only work if the person actually wears them or is conscious, some users forget to wear them or decide not to wear the device [19]. Wearable systems are also vulnerable to the stubbornness and pride of the elderly. This might prevent a person who might require help from actually pressing the alert button.

1.5.2. Cameras

Vision-based techniques are also very common when it comes to fall detection and have been seeing major improvements in the last 5-6 years [14]. A recent technique with a camera is to translate the images to curvature scale space (CSS), which means that an image is transformed into a silhouette. As a silhouette itself can be very noisy and have many local deformations, curve interpolation can be used to reduce a silhouette to a simple form [20]. This is a very valuable technique to detect different positions of a body with low-resolution measurements which can also be used in combination with other sensing techniques. Another more recent technique in vision technology uses infrared, there are even techniques that use both infrared and wearable solutions [21]. Most infrared techniques use the Kinect module since this is a readily available sensor, "The working principle of the Kinect depth sensor is to actively project near infrared spectrum by an infrared projector. When the infrared rays radiate to rough objects, the spectrum is distorted, and form some random reflection spots. Its infrared camera captures these changes in the reflected infrared spectrum." [22]. A large downside of this technique is the fact that it needs a 'base' image of a room (without people in it).

1.5.3. Radar and WiFi

The Doppler effect can be used to detect movement, by using either Continuous Wave Radar [23][24] or WiFi [25]. Machine learning can then be applied to let the system decide whether the movement detected is a fall or not. The main advantage of these techniques is that they do not form any visual image, but only measure movement speed, which is desirable from a privacy standpoint. The major downside of using machine learning is that there is not a lot of data available to train with. One could simulate falling and use that for training, but then the system is only trained for that particular person. Different body shapes and personal habits make it complex to get a system to work universally.

1.5.4. Floor Mounted

Floor mounted sensors can be capacitive [26][27] or work on vibration [28][29]. Vibration sensors can work well on hard falls but not so well on a slow collapse as these do not cause as much vibration. Differentiating between collapses and daily activities can become very complex. Capacitive sensors can detect stationary objects and therefore also people lying on the floor. This is a huge advantage for the capacitive sensor as they are sensitive to all kinds of falls, as long as the person lands on the floor. The disadvantage of capacitive systems is the more complicated hardware required to collect the desired data. This is due to the frequency requirements when building this type of sensor.

2
Global system requirements

The project group was tasked with developing an activity and fall detection system, as described in chapter 1. To realize this project, the system requirements will have to be observed first.

The functional requirements will describe the features that have to be implemented whereas the non-functional requirements will describe the workings and constraints of the system under certain conditions.

2.1. Functional requirements

R1.1 Both slow and fast falls should be detected.
In section 1.4, a fall was defined as coming to the ground as the result of both a sudden change in body position or a slow collapse. The systems must be able to detect both.

R1.2 The system must have alarmed a relevant person within 1 minute after a detected fall
The system should alarm about a fall detection so that the person contacted or notified is always aware of this.

R1.3 System must be operational for 99% of the time
A fall can happen at any given time, so the system should have minimal downtime.

R1.4 Furniture or static objects should not influence the detection process
Every home has furniture and furniture shading is a major obstacle for most sensor types. The system needs to be able to detect a person falling despite there being furniture present in the same room.

R1.5 The system should be scalable
It should be possible to deploy the system in any room and expand the system to multiple rooms or homes within a building.

R1.6 The prototype should have a demonstration mode, showing:

(a) **The location of a person in the room.**

(b) **If a person has fallen.**

This demonstration mode should operate as a showcase where people can walk across the room and see their live location on the screen alongside an indicator to show whether they have fallen.

R1.7 The system should only report falls when one person is present in the room.
When multiple people are present, there is no need for a detection system, as the other person can help.

2.2. Non-functional requirements

R2.1 A fall must be detected with false negative rate lower than 10%
False negatives should be avoided as much as possible, leaving an elderly person on the floor without alarming would mean the entire system has failed.

R2.2 A fall must be detected with a false positive rate lower than 20%
While false positives are less important to avoid than false negatives, calling many false alarms is undesirable and may cause people to act less serious on alarms.

R2.3 The system must not use camera systems for detection.
Due to privacy reasons, the project does not allow using a camera. This includes the use of any form of visual sensor which can be used to reconstruct a recognizable image, e.g. some types of infrared camera's.

R2.4 The system must not use audio recording systems for detection
Due to privacy reasons and the responsibility these recordings would add [30, Recital 51], the decision was made not to use audio systems to detect a fall. This includes any form of sensor that leads to understandable audio recordings.

R2.5 The system must not use devices placed directly on client for detection
Relying on elderly people to always wear a device harms the reliability of the system since forgetting to wear the device might result in a false negative.

R2.6 Complete system should not be noticeable
The system is aimed to be present in the homes of mostly elderly people. Since their feeling of independence should be preserved, the system is allowed to have at most one visible terminal or device for user feedback.

R2.7 The activity and falling detection should not rely on the feedback of a user
The system should be able to detect a fall without the user letting the system know that he or she has fallen, or in other words, the system should be able to detect a fall without the user being conscious.

R2.8 The system should be operational in 30 seconds after powering on
Powering on the system is seen as plugging the system into the power outlet, where operational means that falls and activities can be detected.

The following requirements apply to a room of 4 by 5 meters:

R2.9 A single person should be able to install the system within 4 hours
This includes only the installation of the fall detection system, finishing of the room is not included in this time.

R2.10 The maximum material costs per room (4m x 5m) should be €1.000
Including all the material cost, excluding the labor cost.

R2.11 The maximum operational costs per year for a room (4m x 5m) should be €250
This includes power and maintenance costs.

R2.12 The maximum end-of-life costs for a room (4m x 5m) should €1.000
This includes the removal of the system and processing of the materials.

3
Initial design decision

There are a lot of ways in which a person can fall. This makes it difficult, maybe even impossible, to detect all falls with a single type of sensor. Combining sensors can be a way to get better falling detection. Of course, every type of sensor works differently, thus implementing this can be a very time-consuming task. Another, more preferable option, is to decide on one type of sensor that can detect a large number of falls and optimize this. Covering 90% of all falls using a single type of sensor can end up being much cheaper and less time consuming for the development team. Therefore, a single solution will be selected that fits the requirements set in chapter 2 and this will be based on one of the solutions presented in section 1.5

3.1. Requirements compared to different implementations

In section 1.5 implementations using four different categories of sensors were described:

1. Camera
2. Wearable
3. Radar and WiFi
4. Floor mounted

A solution using a camera or wearable is not possible since they do not meet requirements **R2.3** and **R2.5**. This leaves radar/WiFi and floor mounted as possible options. Requirement **R1.1** states that slow falls (collapses) should also be detected. Radar and WiFi-based systems detect movement speed, meaning it is possible to detect hard falls, but slow falls will be near impossible to detect due to the low movement speed. This leaves a floor mounted sensor as the remaining option. Of this implementation, two kinds exist: Pressure based and vibration based. Again, a vibration based sensor would have a hard time detecting a slow collapse, because this would cause little vibration. Additionally, it could be possible to record audio using vibration sensors, depending on the sensitivity and the sampling frequency. As higher sampling rates and sensitivity are likely desirable for detecting a fall reliably, a vibration sensor could be privacy intruding.

The last option is a pressure based floor sensor. This type of sensor is able to detect a fall as described in section 1.4, and is thus able to detect a fall independent of how someone has come to the ground (i.e. fast or slow). Furthermore, furniture should not cause a problem, as these are static and should, therefore, be able to be filtered out, thus meeting requirement **R1.4**. Additionally, a floor mounted sensor can also be placed under carpet or vinyl, thus meeting requirement **R2.6**, concerning the notability. Finally, this solution is independent of feedback of the user, thus meeting requirement **R2.7**.

Summing up the advantages and disadvantages, it is chosen to use a pressure based floor type sensor system. The specific choice is elaborated in the hardware subsystem report [3].

3.2. Project outline

In section 1.1 the general setup of the project was described, dividing the system into three subsystems. A general overview of the system layers can be seen in figure 3.1. With the initial design decision now made in section 3.1, each subsystem's functionalities can be described in further detail. The subsystems are: **Sensor design and hardware abstraction layer**, **Interface**, and **Fall Detection Algorithm**. Figure 3.2 shows how the three subsystems are connected and what the data flow is.

Sensor design and hardware abstraction layer
The sensor design and hardware abstraction layer subsystem is responsible for reading out the sensor data of the pressure-sensitive floor. The subsystem will not only contain the necessary hardware but the Hardware Abstraction Layer (HAL) as well. The subsystem's responsibility ends at digitizing these signals.

Interface
The interface subsystem is responsible for the communication between the two other subsystems. This subsystem will pass along requirements about the communication methods used between subsystems. Additionally, this subsystem will also interface with the outside world and signal for help when a fall has been detected.

Fall Detection Algorithm
The final subsystem is responsible for the algorithm that is able to detect a fall based on the output of the pressure sensitive floor. The output of the algorithm is linked back to the interface subsystem.

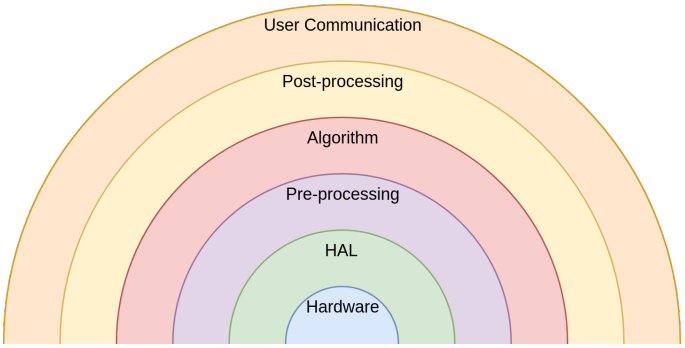

Figure 3.1: System layers

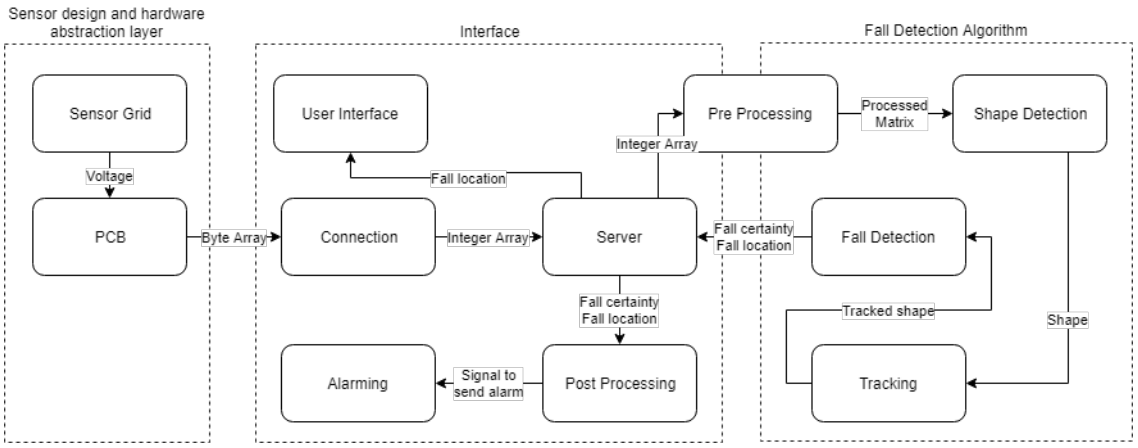

Figure 3.2: Advanced system overview with data flow

4
Interface Introduction

4.1. Overview

From now on this report will discuss the design of the interface module. The interface module provides a server that handles the following tasks:

- **Communication from the hardware module**
  For readability, the sensor design and hardware abstraction layer will be shortened to the hardware module, this module provides a digital representation of analog measured voltage values. The server needs to provide a link to get these values onto the server platform.
- **Pre-processing the raw data from the hardware module**
  The representation provided by the hardware module is not representative of the actual physical layout of the sensor. The server performs pre-processing to turn it into a representative format.
- **Back and forth communication with the processing module**
  For readability, the fall detection algorithm module will be shorted to the processing module. The server sends pre-processed data to this processing module and retrieves the output.
- **Habit tracking**
  The processed data is used to track the client throughout the habitational environment. This tracking is used to check whether the behavior of the client is showing anomalies.
- **Alarming**
  The server decides if there is a need to alarm and starts calling for assistance using autonomous phone calls.
- **Providing a user interface that shows accessible information to the user**
  For presentation and ease of use, the server provides a simple user interface that displays the status of the system and what it is detecting.

A block scheme of the interface module can be seen in figure 4.1.

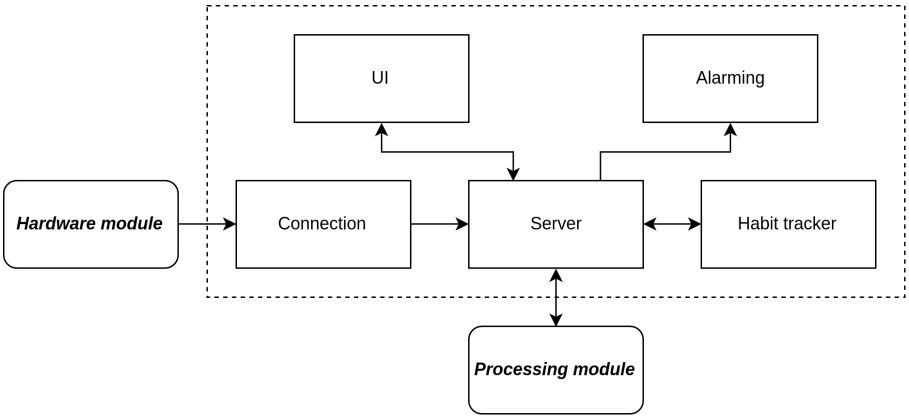

Figure 4.1: Diagram of the interface module

4.1.1. Used hardware

All developed software for the interface module is tested to run on Windows 8, Windows 10 and Ubuntu 18.04. Performance measurements are done on a laptop running Ubuntu 18.04 with an Intel Core i5-8250U processor and 16GB RAM (Asus Zenbook UX410UA-GV354T with expanded RAM).

4.2. Subsystem Requirements

Because the interface module is connected to both the hardware and processing modules, it is important to set requirements for the interfacing between these modules. Furthermore, there are requirements for each component of the interface as well, such as the habit tracking and alarming systems. Some subsystem requirements follow from the global system requirements as described in chapter 2.
This section will list all subsystem requirements with a short explanation when needed, divided into subsections.

4.2.1. Server

SR1.1 The server must have an internet connection
Internet is needed to make programmable phone calls

SR1.2 The server has to comply with requirements R1.3, R2.7 and R2.8
This means that the server has to be on 99% of the time, work autonomously and have a startup time of less than 30 seconds.

4.2.2. Interfacing with the hardware module

SR2.1 The connection needs to be invisible
Because of **R2.6**, the visible terminal is reserved for the user interface. This means that the entire connection needs to be wireless or consist of cables under the floor.

SR2.2 The connection needs to be stable
According to **R1.3**, the system needs to be running 99% of the time which also means the connection needs to be running and sending/receiving data 99% of the time.

SR2.3 The interface needs to support at least 25 connections
Because the hardware module has a modular design, the interface between the hardware and software needs to be able to support multiple connections at once (**R1.5**), since multiple modules of hardware will be sending data to the server. A decently sized apartment would need at least 25 connections.

SR2.4 Sending data from one hardware module should have a maximum delay of 5 seconds
Less delay provides a quicker response to a fall, but low latency is not needed for the processing algorithms.

SR2.5 Interface compatible with Python
The server will run on Python code so this connection needs to be able to interface with Python.

4.2.3. Pre-processing

SR3.1 Every input signal needs to be decoded and transformed
The processing module expects a matrix representing the room, so the byte signal input needs to be converted to a matrix and transformed so that it represents the physical grid.

SR3.2 Multiple hardware modules need to be combined into one representation
The processing can only perform its algorithm on one matrix. If multiple hardware modules exist, they all should be combined internally into one matrix.

SR3.3 The information sent to the processing module has to contain a matrix with a timestamp
A timestamp is required by the processing module for stability and checking system performance.

SR3.4 Pre-processing is allowed to increase the total execution time by a maximum of 5%
Execution time is defined as the time it takes to fully process the data from being received at the server to all processing being done.

4.2.4. Post-processing

SR4.1 Post-processing is allowed to increase the total execution time by a maximum of 5%
Execution time is defined as the time it takes to fully process the data from being received at the server to all processing being done.

SR4.2 The server has to decide, based on R1.2, if there is need to call for alarm
The server has a 1-minute window to decide if a person has fallen, this decision is based on the outputs of the processing module and habit tracker.

SR4.3 Alarming should be done by calling a prioritized list of phone numbers
This list can contain any phone number given by the client.

SR4.4 The alarmed person should be able to confirm or decline the request to come help
The server needs to know whether it should continue to call the next in line on the prioritized list.

SR4.5 The habit tracker should return a percentage representing how normal the client is behaving
This percentage can be used as an indicator for the alarming decision making.

SR4.6 False positive and false negative rates should comply with R2.2 and R2.1

4.2.5. User Interface

SR5.1 The UI needs to have a calibration button for the processing module
The user needs to be able to tell the system that, for example, some furniture moved and that it needs to re-calibrate.

SR5.2 The UI should display the location of the person walking on the grid (R1.6)

SR5.3 The UI should display whether a person has fallen or not (R1.6)

5
Hardware communication

The interfacing between the hardware and the server is the first step the sensor data has to pass through. The endpoint of the hardware module is a PCB with digital sensor data and this data needs to reach the server.

5.1. Design Decisions

The requirements set up in section 4.2 are important to keep in mind when deciding on a communication method. Most importantly **SR2.1**, **SR2.2**, **SR2.3**, **SR2.4** and **SR2.5**. The first design decision that needs to be made is whether the communication should be wired or wireless. It was decided that the communication should be wireless for several reasons:

- Wireless communication automatically fulfills **SR2.1** while keeping wires invisible can prove to be difficult.
- Python (**SR2.5**) has support for most wireless implementations.
- It is easier to scale wireless systems because there is no need to consider physical inputs on the server.
- The project group has more experience with wireless communication when it comes to a large amount of connections, and will be able to deliver this faster.

Stability and delay start playing a larger role when it comes to wireless communication, but an effective protocol will be stable and fast enough for this project. A few wireless communication methods that have strong Python support are Bluetooth, ZigBee, and WiFi.
All three of these methods can be implemented on the hardware, which means they are all valid choices. Out of these three, it was decided to use WiFi for the following reasons:

- In this day and age WiFi networks can be found anywhere, this makes it easier to set up because the system can connect to an already existing home network.
- While ZigBee works very similar to WiFi and can prove to be more stable, it is a much costlier method to implement
- WiFi (and ZigBee) have better support for a large number of connections than Bluetooth which is important for **SR2.3**
- WiFi protocols are more secure than Bluetooth ones [31], this makes it easier to secure private information in the future

To implement WiFi communication, the MQTT protocol [32] will be used. This protocol has a large amount of documentation, allows for modularity using "topics" and works well with Python.
MQTT also supports a large number of connections which fulfills **SR2.3**. Furthermore, it is a robust protocol that allows tweaking of the Quality of Service to improve the connection speed and vice versa.

5.2. Detailed description

The MQTT protocol works by starting an MQTT Broker on a local network. A diagram of the MQTT Broker interaction can be seen in figure 5.2. Clients can connect to this broker and subscribe to "topics" that are relevant to them. The clients can publish data to these topics and the MQTT Broker will then send this data to all clients subscribed to this topic. This means that, in the case of a modular system, each hardware module can publish data to a separate topic while the server is subscribed to every topic. This way the server can identify which hardware module is communicating by looking at the topic name. Because the server knows which hardware module is communicating, it can effectively combine the information sent by each module during the pre-processing. An example of how topics are assigned can be seen in figure 5.1. As can be seen from the figure, using these topic names as identifiers can be an effective way to pinpoint the location of each module.

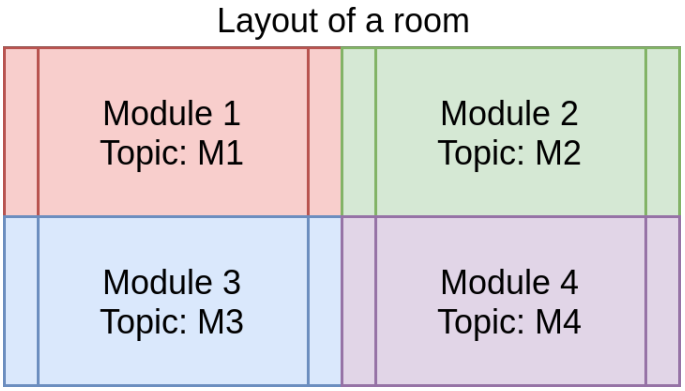

Figure 5.1: Schematic view of a room covered in 4 hardware modules

The hardware will be communicating with the MQTT Broker using ESP8266 modules that are hardcoded to publish to a specific topic. Each message sent will be 1024 bytes and contain the entire sensor readout of a module. The communication method used by the broker is a TCP protocol and has three Quality of Service levels:

- QoS level 0: A packet is received at most once.
- QoS level 1: A packet is received at least once.
- QoS level 2: A packet is received exactly once.

Because only 99% of the packets need to arrive according to **SR2.2**, the broker is set to QoS level 0 for maximal throughput. The server is constantly listening to the MQTT Broker for new messages and as soon as a new message arrives it will be sent to the server and placed into a Queue for pre-processing. This Queue is used to separate the communication and processing parts of the software so that they can run simultaneously.

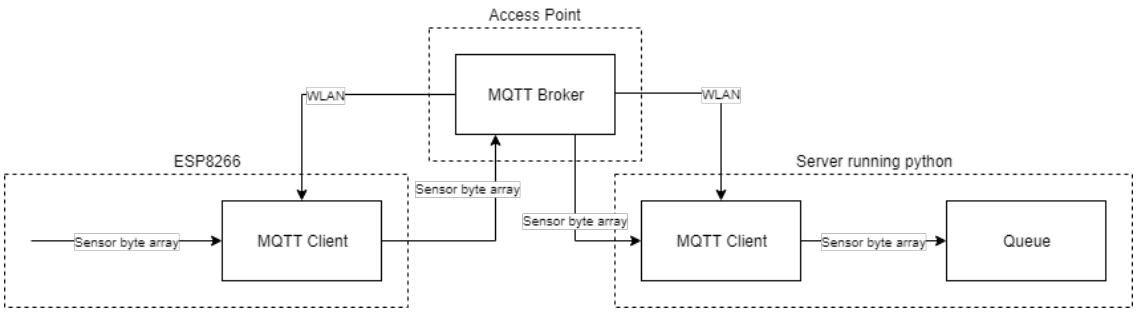

Figure 5.2: Block scheme for the connection using MQTT

5.3. Evaluation

To validate the requirements, datasheets of the protocol were used and a few testing scenarios were created:

- **SR2.1**:
  Because a wireless communication method was chosen, the connection is by default invisible.
- **SR2.2**:
  By sending a certain number of packets and checking how many are received at the server it is possible to validate requirement **SR2.2**. The ESP8266 was programmed to send exactly 5000 packets to the broker and the server was checking how many packets it received from the broker. This was done multiple times over 2 hours and almost every attempt all 5000 packets were successfully received. A few packets were lost during attempts due to the cause of short disconnects, but the connection still retained an uptime higher than 99% (≈30 seconds of disconnects over 2 hours), and while the connection was up, 100% of the packets were received.
- **SR2.3**:
  To fully test if the amount of connections allowed is high enough, too many ESP modules are required, however, the specifications of the MQTT Broker used state that the default maximum amount of connections is 1024 and this can even be increased if needed [33].
- **SR2.4**:
  Using the graphical tool MQTT.fx, it is possible to inspect the timestamp of each message sent through the broker. By looking at the serial monitor of the ESP and comparing the two timestamps it is possible to get a good estimate of the delay. A test setup is made that sends 5 packets from the ESP and prints the timestamp of the 5^{th} packet to avoid the initial connecting delays. Comparing these timestamps shows a difference of 65 ms which is the delay between printing on the serial monitor and receiving the message. Printing to the serial monitor is included in this measurement, and this has a small delay as well. This means the delay is less than 65 ms and fulfills the requirement.
- **SR2.5**:
  MQTT has python support using the Paho MQTT package [34].

6
Pre-processing

The interface module has the task to deliver the sensor data from the hardware module to the processing module. The processing module expects this data in a format that is representative of the actual physical sensor. The raw data from the hardware module does not comply with this demand, and therefore the interface module needs to do pre-processing on the raw data. This chapter will discuss the pre-processing and the interfacing with the processing module.

6.1. Design Decisions

In this section, it will be discussed how the requirements for the pre-processing are planned to be met, and why those choices have been made.

- Requirement **SR3.1** states that the input signal needs to be decoded and transformed into a matrix that represents the physical grid. To comply with this requirement, there is not much to decide upon. The raw data is delivered through the MQTT protocol, as discussed in chapter 5, as one binary number for every readout of one hardware module. The processing module needs a matrix of unsigned integers, so it is converted into exactly that.
- Requirement **SR3.2** states that multiple hardware modules need to be combined into one matrix. It is decided to first perform the decoding and conversion to a matrix, after that all matrices are concatenated together to form the total representation.
- Requirement **SR3.3** is a clear cut demand and requires no decision making.
- To comply with **SR3.4**, the pre-processing cannot take more than 5% of the total execution time. Performing the pre-processing in a sequence after each packet is received from the interface with the hardware module might have too much impact on the execution time. Instead, it is decided to perform the pre-processing in parallel. From the hardware interface, the data is put into a queue, from which the pre-processing module pulls the data and does its job in the background while other processes can continue. This makes the pre-processing, in theory, not affect the execution time at all. This holds as long as the pre-processing in the background does not take longer than the processing done in the processing module, and the CPU does not run into its limitations, both of which are not very likely to happen.

Lastly, it is decided to perform all pre-processing on the same hardware as all other software. This makes interfacing with other modules simple.

6.2. Detailed description

The main design of the pre-processing is best expressed in a block scheme, as can be seen in figure 6.1. First of all, data is pulled from the queue, which is formatted as one binary number at this moment. This is converted into a matrix of unsigned integers with a dimension of 16 by 32 (respectively width, height), since this is the actual dimension of one hardware module. However, as discussed in [3], the grid of the hardware module is not square but skewed. To obtain a good representation of the hardware module, the matrix also needs to be skewed. This is done by creating a matrix of size 160 by 320 (ten times the original size) and placing the data from the original matrix into the larger matrix at the right location.

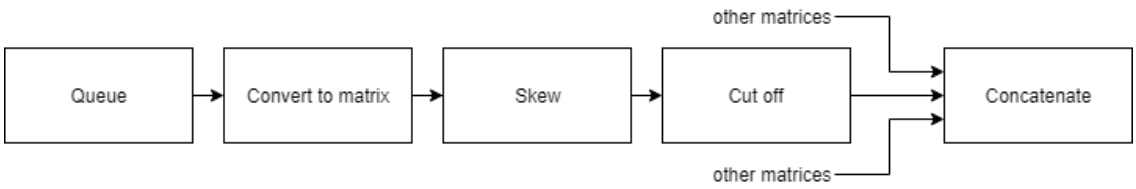

Figure 6.1: Block scheme for the pre-processing

The right location in the new matrix is determined by creating a mapping function. The design of the grid with the known dimensions is shown in figure 6.2, for more on the hardware design see [3]. From these dimensions, mapping functions 6.1 and 6.2 can be made, with (x, y) defined as the original coordinates and $(0,0)$ defined as the measurement point at the bottom left.

$$x_{map} = \frac{4 \cdot x}{\tan(30)} \approx 7 \cdot x \tag{6.1}$$

$$y_{map} = 8 \cdot y + 4 \cdot x \tag{6.2}$$

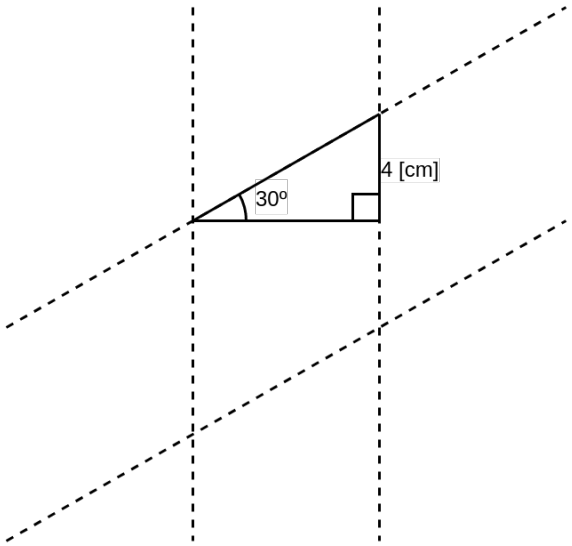

Figure 6.2: Grid dimensions

These mapping functions return a value in centimeter. The pixel size (distance between two elements in a matrix) is one centimeter with the large matrix being ten times the original size, the coordinate in centimeter is the same as the coordinate in the matrix, rounded off to the nearest integer. After skewing, the result is not yet satisfactory. Some of the measurement points at the bottom and top of the skewed matrix are non-existent on the prototype. To fix this, the non-existent points are cut-off, slightly decreasing the height of the larger matrix. The result can be seen in figure 6.3. The image on the left shows the larger matrix when no skew has been done to the data points, and the image on the right shows the final matrix after the skewing and the cut-off. The data points in the matrix are fictional and only there to visualize the pre-processing.

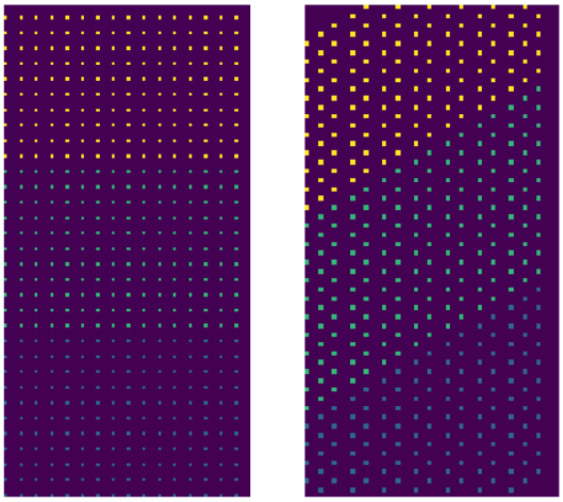

Figure 6.3: Left: non-skewed matrix, right: skewed and cut-off matrix

As mentioned before, the whole system is built upon modularity, and therefore the pre-processing needs to take into account the possibility that there are multiple hardware modules. This means that the pre-processing has to concatenate all matrices for all hardware modules together into one matrix. As discussed in 5.2, the topic that the hardware module publishes on represents the placement of this module and allows for distinguishing between the modules. The placement of the hardware module is used to determine how the matrices should be concatenated (in what order, on top or side by side, etc).

Lastly, the current time with an accuracy of milliseconds is added. The matrix combined with this timestamp is fed into a queue, where the processing module can pull from.

6.3. Evaluation

Three out of the four requirements are purely functional requirements. This means that validating these is a matter of crossing off whether they are implemented or not. The functional requirements are **SR3.1**, **SR3.2** and **SR3.3**.

All three of these functionalities have been implemented and are working as expected. Requirement **SR3.4** needs testing to validate. Testing the impact on execution time is difficult to assess when the pre-processing happens in parallel to the rest of the software. Turning off pre-processing to see the difference is not a viable solution since all other modules expect a working pre-processing module and will break if it is not working. However, the pre-processing module reads the queue faster than it is filled. This means that the pre-processing is faster than the update frequency of the hardware module and therefore does not impact the execution time at all. So at this moment, the requirement is fulfilled, but this might change in the event of a faster hardware module being developed. In this case, it is possible to start to do the pre-processing in two physical processes, doubling the effective speed at which incoming data can be handled.

7
Habit Tracker

The goal during the design of the habit tracker is to create an algorithm that determines how normal the client is behaving on a given day. The habit tracker is used as an addition to the output of the processing module to get greater accuracy on the fall detection. Also, the alarming module might decide to alarm even when no fall is detected in the case of many anomalies in the habit tracker.

7.1. Design Decisions

The habit tracker has only one requirement, **SR4.5**, which states that it should return a percentage representing how normal the client is behaving, the habit deviation. The habit tracker is added as one of the last elements in the system and therefore has to adapt to be able to interface with the other modules. It is also desirable to make sure the habit tracker has no impact on the performance of other modules. At the very least it should comply with requirement **SR4.1** as it is part of the post-processing, but preferably less impact.

To achieve the only requirement, it is decided to create a profile that represents the activity per hour in a day. This profile is compared to another logged profile that represents a recent average. More difference between both profiles would mean a higher habit deviation, less difference means low habit deviation. The complete habit tracking is kept in a separate process running in the background, meaning it should not impact the performance of other processes as long as the CPU is powerful enough.

7.2. Detailed description

The processing module returns (among others) the location of the last known footstep detected on the sensor grid. This location is put into shared memory that the habit tracker has access to. The habit tracker reads the location and compares it with the previously read coordinate. If these locations do not match, the habit tracker knows that there has been activity.

7.2.1. The profile

The profile is an array of size 24, one element representing one hour in a day. The elements in the array are made using the same type of matrix used to represent the sensor array (matrix of unsigned integers, 8 bits). The dimensions of the profile are chosen in such a way that one pixel (which is one element in the matrix) is 1 by 1 meter. The log profile as mentioned in the previous section has the same dimensions. One matrix in the profile from now on will be called a frame.

7.2.2. The algorithm

The algorithm is triggered when activity is measured. The coordinate of the footstep is translated into a coordinate in the profile using equation 7.1.

$$\text{Profile coordinate} = \text{Footstep coordinate} \cdot \frac{\text{Profile dimension}}{\text{Sensor matrix dimension}} \tag{7.1}$$

After this, the current hour is determined, which corresponds to a frame within the profile. In this frame, the profile coordinate as determined by equation 7.1 is increased with 1. In case of overflow (which is at value 255), no value is added. When a new day begins, or in other words, when the current hour is zero, the log adds the profile of the last day to the log profile. This is done by looping over every pixel in every frame. The new value for the pixel is determined by taking the outcome of equation 7.3

times the value stored in the log, and adding the outcome of equation 7.2 times the value in the profile of the last day.

$$\text{Profile scalar} = \frac{\max\left(10 - N, 1\right)}{10}, N = \text{Number of times log updated} \tag{7.2}$$

$$\text{Log scalar} = 1 - \text{Profile scalar} \tag{7.3}$$

The calculation of the habit deviation is done by, once again looping over every pixel in every frame in the current profile and taking the difference between this value and the value stored in the log for this pixel. All these differences are added up and divided by the maximum possible difference. The resulting habit deviation is stored as a 64-bit floating point number in shared memory so that other modules can access it.

7.3. Evaluation

A test setup has been created to validate the habit tracker. The ideal test setup would be to run the habit tracker for at least 10 days in a full room while recording the activity with a camera for comparison. This is not feasible within this project, so instead of this, a small scale test setup is made. For this, instead of using a complete room, one hardware module is used. The dimensions of the profile are multiplied by 10, creating a pixel size of 10 by 10 centimeter instead of 1 by 1 meter. Instead of using a frame for every hour (24 frames), two frames are used. The current frame is determined by taking the current minute modulo two. Instead of updating the pixel by adding a 1 every time activity is measured, the pixel is immediately set to 255, which is the maximum possible value. The result is a small scale and a much faster version of the original algorithm.

In this setup, a person walked around in circles for a few minutes. This led to the log and current frame looking like figure 7.1. The habit deviation remained low while walking roughly the same pattern. After this, the person started walking in a much smaller circle. This led to the current profile being different from the log that was created before, as can be seen in figure 7.2. The habit deviation was much higher in this scenario.

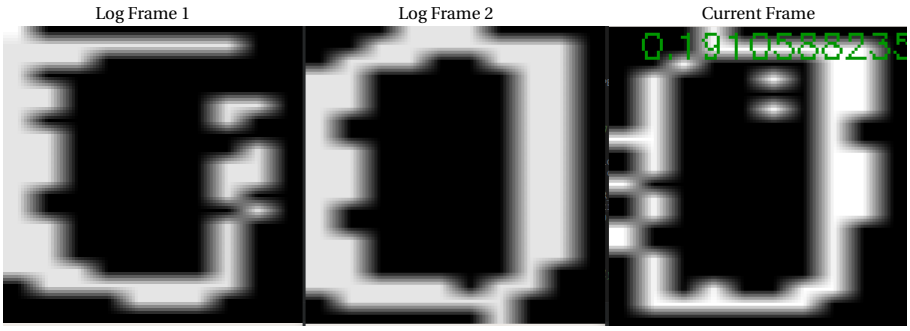

Figure 7.1: Low habit deviation (value in green, 19%)

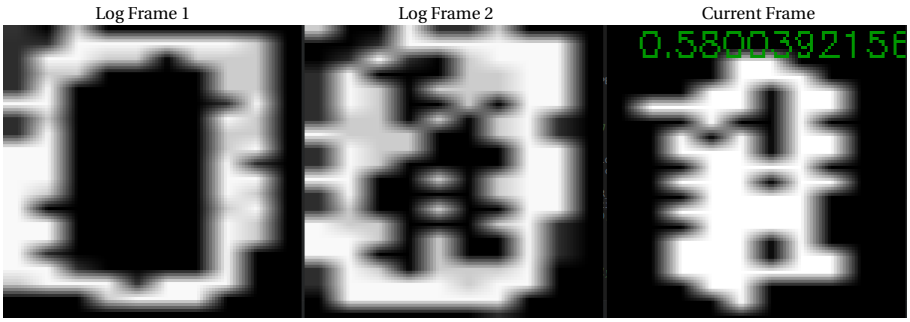

Figure 7.2: High habit deviation (value in green, 58%)

For now, the conclusion is that the habit tracker is a proof of concept. It works on a small scale but would need a larger scale test to calibrate the algorithm and to decide what percentage should be considered normal, and what percentage should be considered abnormal.

As mentioned before, the habit tracker works in a separate process in the background. It updates a shared memory value that is always defined and therefore no other module has to wait for the habit tracking to be done. This means that it has no effect on the execution time as long as the laptop that the software is running on does not run into CPU limitations. This was not the case on the laptop used.

8
Alarming

The alarming module functions as the post-processing for the processing module and handles the alarming. It implements a time-based comparison of the processing results to come to a better conclusion about whether there is a need to alarm or not. When alarming is required it makes text-to-speech phone calls to call for assistance and to make sure the user is not left helpless on the floor.

8.1. Design Decisions

The alarming module has to fulfill almost all the requirements mentioned in section 4.2.4 (**SR4.1**, **SR4.2**, **SR4.3**, **SR4.4**, and **SR4.6**). It was decided to look at the average output of the processing module in a certain time window to determine whether to call for alarm or not. This makes the alarming module an addition to the complete decision making. To fulfill requirements **SR4.3** and **SR4.4** the choice was made to use a programmable voice API called Twilio [35]. Twilio was chosen for the following reasons:

- Twilio has support for Python which means that no additional program is needed and it can be added to the main fall detection software.
- The Twilio voice API allows for the gathering of button inputs which allows for the implementation of **SR4.4**.
- Twilio has a lot of documentation which makes it easier to troubleshoot while writing the software.
- Twilio is free to use (for a limited time) unlike competitors like Skype. This means that there is no additional cost during prototyping and testing.
- A lot of competitors of Twilio only provide access to programmable text messages and this is not ideal. A programmable text message can be used to call for alarm but it is more difficult to get a confirmation message required for **SR4.4**.

8.2. Detailed description

An algorithm is implemented in the alarming module that checks the last 30 seconds of data received from the processing module. By looking at a time window instead of one data point it is possible to calculate the average certainty of a fall over the last 30 seconds. The output of the processing module is stored by appending it to an array. The stored data contains a timestamp, so every data point that has a timestamp older than 30 seconds is discarded. If the average certainty output exceeds a preset threshold, an alarm signal is sent out. This way, with a proper threshold, it is possible to filter out occasions where, for example, a person sits down on the floor to pick something up. While the tuning of this threshold is dependant on the processing module's results, reducing the threshold will reduce the false negative rate and increase the false positive rate. Increasing the threshold will have the opposite effect. This can be used to fine tune for requirement **SR4.6**.

As mentioned earlier, Twilio is used for making programmable phone calls. Twilio provides a voice API that can be programmed to say anything. After it is decided to alarm, the flowchart in figure 8.1 is followed.

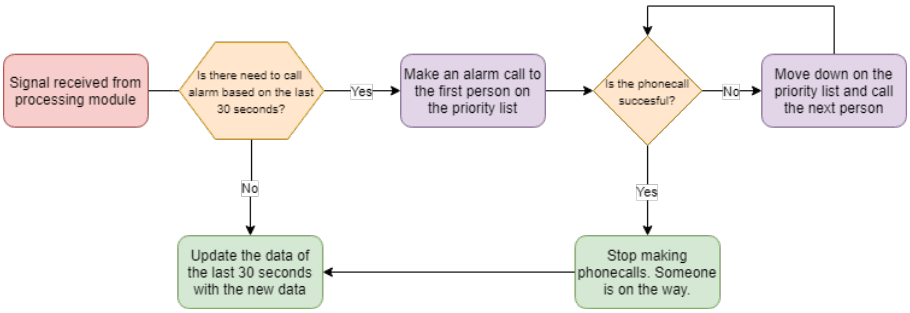

Figure 8.1: Flowchart of the alarming

When calling it is possible to use the personal name of the client to make it clear to the person that is being called who has fallen. Creating the voice and flow of the phone calls is done by using the Python API that Twilio provides. When run, this Python code outputs in a language called TwiML, which is a markup language created by Twilio. This TwiML code is posted to a public website using the Flask API [36]. The website is hosted using ngrok [37], which allows for exposing a local web server to the internet. Input on the phone's keyboard is used to confirm or deny that the phone call was successful. Twilio's server handles the input on the phone by posting it to the same public website that is hosted by the alarming module. Flask is used to get these posts, from which the relevant data is filtered out (which is what buttons are pressed). The called person is asked to press 1 to confirm or 2 to deny.

8.3. Evaluation

Requirement **SR4.2**, **SR4.3**, and **SR4.4** are all functional requirements that have been implemented as described in section 8.2. Requirement **SR4.1** and **SR4.6** need testing to be validated. To test the execution time of the decision making equation 8.1 is used.

$$\text{Execution time ratio} = \frac{\text{Max(post-processing execution time)}}{\text{total execution time}} \cdot 100\% = \frac{0.053\text{ms}}{57\text{ms}} \cdot 100\% \approx 0.09\% \quad (8.1)$$

To determine the post-processing execution time and total execution time, timers are placed within the software. The software is started and left on for several minutes to make sure that the times are accurate. As can be seen from the equation, the impact of the post-processing on the total execution time is approximately 0.09% which is much lower than the demanded 5%.
To validate the false positive and false negative requirements there is a need to assume that the processing module has already validated the false positive and false negative requirements for one measurement instance. To test the effectiveness of the 30-second window, a test subject walks onto the fall detection floor simulating multiple types of falling. The test subject also walks onto the fall detection floor simulating actions that do not include falling but may be detected as falling. The results from these measurements can be seen in table 8.1 and 8.2

Measurement Type	Processing result	Post-Processing result
Person falling suddenly	True Positive	True Positive (100%)
Person falling suddenly and trying to get up	True Positive	True Positive (89%)
Person slowly collapsing	True Positive	True Positive (100%)
Person falling and lying on their side 4	True Positive	True Positive (100%)

Table 8.1: Results of measurements where a person has fallen in the last 30 seconds (threshold set to 66%)

Measurement Type	Processing result	Post-Processing result
Person walking	True Negative	True Negative (12%)
Person crawling	False Positive	True Negative (23%)
Person sitting down	False Positive	True Negative (47%)
Person kneeling	False Positive	True Negative (0%)
Person sitting and standing up	False Positive	True Negative (10%)
Person lying down	False Positive	False Positive (100%)
Person lying down and standing up	False Positive	True Negative (31%)
Person squatting	True Negative	True Negative (30%)
Person picking up a pot and placing down	False Positive	True Negative (19%)
Person sitting on a chair	True Negative	True Negative (18%)

Table 8.2: Results of measurements where a person has not fallen in the last 30 seconds (threshold set to 66%)

As can be seen from the measurement results, the added post-processing has no impact on detecting actual falls (8.1), which means there is no change in the false negative rate. It does have a very large effect on filtering out actions that should not count as falls (8.2), which means that the false positive rate is reduced. The only scenario where the post-processing is unable to filter out an incorrect result from the processing module is the case of a person lying down. This scenario gives the same result as a person slowly collapsing, and thus cannot be filtered out in any way.

9
User Interface

The user interface is used to show clear and concise information about the system. This means that it will only show information that is directly relevant for presentation purposes and hide all other information to improve readability and understanding.

9.1. Design Decisions

Aside from the user interface requirements described in section 4.2, the user interface also needs to be integrated with the main program. This means that it needs to support multiple processes running simultaneously and be able to receive information from these. Python has a large number of packages that allow for the building of a graphical user interface, the most popular and most supported of these are Tkinter, PyQT and Kivy. Any of these packages can be used to write a functional user interface, however, based on user reviews[38][39], Tkinter seems like an unsuitable solution because it can not handle multiple processes very well and this is a very important property of the system. In the end, it was decided to use Kivy for the following reasons:

- Kivy is known for its cross-platform compatibility which allows it to run on tablets and raspberry pi. This gives the ability to expand the user interface even more in the future.
- A large part of the Kivy code is written in a different, easy to understand, language. This means that there will be less python code that needs to be integrated within the main system. Usually, using a separate language is a downside but keeping UI elements isolated is generally a good practice.
- Kivy makes it easier to design an aesthetically pleasing UI compared to PyQT, which in turn makes it easier to design a clear and readable user interface.

It was also decided to present the more important information, such as if a fall has been detected and the location of a person (**SR5.2** and **SR5.3**) as clear as possible and to keep the less important functionalities hidden in a settings panel (**SR5.1**)

9.2. Detailed description

A Python-based Kivy user interface can be written as a combination of two programming languages, Python and KV [40]. This KV language is a language designed by the Kivy developers and can be used to separate functionality and layout code from each other. For example, in KV language it is written that a button should be placed in the top left corner and display a certain text, while in Python code it is written what pressing the button should do. This separation makes it so that tweaking the layout and location of things in the user interface does not affect on the main system's functionality.
The user interface is designed as shown in figure 9.1 and 9.2. An interface with the important components like user location and alarm status is supported by a smaller, more system-oriented, settings panel.

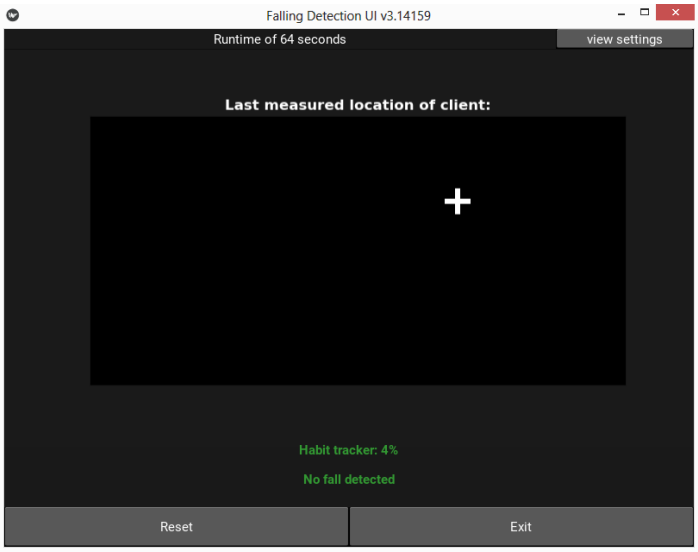

Figure 9.1: Main window of the user interface while running

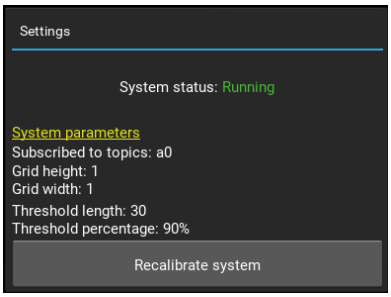

Figure 9.2: Settings window of the user interface while running

The main interface contains:

- A start/reset button to start or reset the user interface.
- A display of the last seen user location, this information is gathered from the processing module.
- A green or red text that displays whether a person has fallen or not sent by the alarming subsystem.
- A green or red text that displays the current habit deviation percentage gathered from the habit tracker.
- A run-time display at the top of the user interface
- A view settings button to open the more advanced settings panel displayed in figure 9.2.
- An exit button to close the entire program.

The settings window contains:

- A display of the current system status which can be either Standby, Running or Calibrating.
- Some system parameters that are relevant to the user interface (for developers).
- A re-calibration button that sends a signal to the processing module, which then allows the system to re-calibrate in case, for example, furniture moved in the room.

9.3. Evaluation

The requirements for the user interface (**SR5.1**, **SR5.2**, and **SR5.3**) are all functional requirements and have been implemented. Location and information whether a person has fallen are displayed in the main window and calibration can be done from the settings panel. These have all been tested by running the main program and checking if the user interface and program are connected.

In theory, the current user interface could run on a tablet or panel in the living room of an elderly person and serve as a simple display, however for this prototype it still runs on a laptop as a demo version.

10 Discussion and Conclusion

10.1. System Evaluation

Section 4.2 described two requirements that the server has to uphold (**SR1.1** and **SR1.2**). Because the prototype is running on a laptop that is connected to the internet, requirement **SR1.1** is automatically fulfilled. Every component of the interface works without any user input, this means that the system is autonomous and satisfies requirement **R2.7**. Starting up the system from scratch with all features enabled takes 3 seconds and the prototype is always running during a demo or testing, this satisfies requirement **R2.8** and **R1.3**. The entire interface subsystem now fulfills all the requirements set for this project. Additionally, all software code is written using the Google style documentation. Because of this, any outside source should be able to follow and understand how the software works.

10.2. Conclusion

In this report, an interface between hardware is designed and implemented. Multiple viable options for this interface were considered from which the WiFi protocol MQTT was ultimately decided as the best option due to its high compatibility with Python, excellent scalability, while the implementation would be relatively low cost. It proved to be stable and fast enough to meet all requirements.

The decision to alarm or not is based on averaging the processing module output for a time window of 30 seconds. This leads to a lower false positive rate compared to looking at the processing module output instantaneously. The execution time performance of this part remained well below the boundary set in the requirements. Alarming is done by using Twilio to make automated text-to-speech phone calls, providing all the necessary functionalities.

A habit tracker proof of concept is provided that checks the daily activities of the client for anomalies compared to a logged average activity profile of past days. This concept is tested in a small scale scenario resulting in promising results that need additional larger scale testing to fully verify. The habit tracker meets the requirements when tested in the small scale scenario.

The user interface has been implemented and works as a visualization for demo purposes. It has been designed to only show simple information that is needed to display the system functionalities, since the processing algorithm was deemed to be too confusing to understand during a demo.

10.3. Future work and recommendations

The time scheduled for the BAP is enough to create a proof of concept and prototype system, however, many functionalities could be added or improved upon if this project were continued in the future. Currently, the habit tracker is a proof of concept but with more testing, this could be iterated upon to have more functionalities like checking if a user is visiting the bathroom an abnormal amount of times or walking strangely because they have an injury. These functionalities could be integrated into the alarming system to provide a much safer habitational environment for elderly people.

As seen from the results, currently, the system does not inform a user that has fallen that someone is on the way to help them. Furthermore, the system also has trouble detecting a person lying down and will call for alarm when this happens. These two problems could be solved by using supplementary hardware like loudspeakers and additional sensors that would greatly enhance the current fall detection.

The current system still uses some prototype hardware and software that could be improved, like the web server used for alarming is set up using ngrok and the MQTT broker runs on a raspberry pi. In

the future, these components of the system could be improved and all be hosted on the same hardware, however, a powerful CPU would be needed for this.

Lastly, the user interface should be integrated into the habitational environment so the elderly people can interact with the system and see what it is doing. This way they can see what is measured by the system and, for example, select that they do not want to have their habits tracked since it might invade their privacy too much. This user interface could also be used as the supplementary hardware to alert a fallen person that someone is on the way. The user interface also needs an override button, this button can be pressed when an elderly person has fallen to call for a manual alarm in case a False Negative situation happens.

11
Global System Evaluation

As can be read from the three subsystem reports, most, but not all requirements have been met. The current system is able to detect falls when a single person is using the hardware, which means that it can be used in many elderly homes already. All three reports mentioned improvements that could be made to this system in the future, and with these improvements the current system would become usable in almost all elderly homes. However, the goal set within the Bachelor Graduation Project was to detect the fall of a single person, which is what the current system is able to do. Therefore it can be concluded that a fall and activity detection system using a pressure sensitive floor can be considered a feasible solution to detect falls amongst elderly.

Bibliography

[1] "Fall fact sheet," 2018. [Online]. Available: www.aging.com/falls-fact-sheet/

[2] S. S. Khan and J. Hoey, "Review of fall detection techniques: A data availability perspective," *Medical Engineering & Physics*, vol. 39, pp. 12 – 22, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1350453316302600

[3] H. Kruissen and B. den Ouden, "Activity and fall detection in the habitational environment: subsystem sensor design and hardware abstraction layer," June 2019.

[4] S. Delfos and E. Granneman, "Activity and fall detection in the habitational environment: subsystem interface," June 2019.

[5] S. Falkena and I. Cornelis, "Activity and fall detection in the habitational environment: subsystem fall detection algorithm," June 2019.

[6] "Centraal bureau statistiek datasheet," 2017. [Online]. Available: https://opendata.cbs.nl/statline/#/CBS/nl/dataset/7052_95/table?ts=1556265088672

[7] WHO, "Falls," 2018. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/falls

[8] W. H. Organisation, "Who global report on falls prevention in older age," 2007. [Online]. Available: https://www.who.int/violence_injury_prevention/other_injury/falls/en/

[9] B. Health, "Bone fractures," Vicoria State Government, 2019. [Online]. Available: https://www.betterhealth.vic.gov.au/health/conditionsandtreatments/bone-fractures

[10] J. Fleming and C. Brayne, "Inability to get up after falling, subsequent time on floor, and summoning help: prospective cohort study in people over 90," *BMJ*, vol. 337, 2008. [Online]. Available: https://www.bmj.com/content/337/bmj.a2227

[11] N. Zorgautoriteit, "Monitor zorg voor ouderen 2018," 2018. [Online]. Available: www.rijksoverheid.nl/documenten/rapporten/2018/04/19/monitor-zorg-voor-ouderen-2018

[12] N. Noury, A. Fleury, P. Rumeau, A. K. Bourke, G. O. Laighin, V. Rialle, and J. E. Lundy, "Fall detection - principles and methods," in *2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug 2007, pp. 1663–1666.

[13] M. GIBSON, "The prevention of falls in later life : A report of the kellogg international work group on the prevention of falls by the elderly," *Dan Med Bull*, vol. 34, no. 4, pp. 1–24, 1987. [Online]. Available: https://ci.nii.ac.jp/naid/10027272265/en/

[14] Y. Z. Tao Xu and J. Zhu, "New advances and challenges of fall detection systems: A survey," March 2018. [Online]. Available: https://www.mdpi.com/2076-3417/8/3/418/pdf

[15] "Automatic fall detection," 2010. [Online]. Available: https://www.lifeline.philips.com/medical-alert-systems/fall-detection.html

[16] D. Yacchirema, J. S. de Puga, C. Palau, and M. Esteve, "Fall detection system for elderly people using iot and big data," *Procedia Computer Science*, vol. 130, pp. 603 – 610, 2018, the 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018) / The 8th International Conference on Sustainable Energy Information Technology (SEIT-2018) / Affiliated Workshops. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050918304721

[17] SisFall, "Sisfall dataset," 2016. [Online]. Available: http://sistemic.udea.edu.co/en/investigacion/proyectos/english-falls/

[18] J. K. Lee, S. N. Robinovitch, and E. J. Park, "Inertial sensing-based pre-impact detection of falls involving near-fall scenarios," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 23, no. 2, pp. 258–266, March 2015.

[19] C. Lin, P. Lin, P. Lu, G. Hsieh, W. Lee, and R. Lee, "A healthcare integration system for disease assessment and safety monitoring of dementia patients," *IEEE Transactions on Information Technology in Biomedicine*, vol. 12, no. 5, pp. 579–586, Sep. 2008.

[20] X. Ma, H. Wang, B. Xue, M. Zhou, B. Ji, and Y. Li, "Depth-based human fall detection via shape features and improved extreme learning machine," *IEEE Journal of Biomedical and Health Informatics*, vol. 18, no. 6, pp. 1915–1922, Nov 2014.

[21] N. Noury, T. Herve, V. Rialle, G. Virone, E. Mercier, G. Morey, A. Moro, and T. Porcheron, "Monitoring behavior in home using a smart fall sensor and position sensors," in *1st Annual International IEEE-EMBS Special Topic Conference on Microtechnologies in Medicine and Biology. Proceedings (Cat. No.00EX451)*, Oct 2000, pp. 607–610.

[22] L. Yang, Y. Ren, and W. Zhang, "3d depth image analysis for indoor fall detection of elderly people," *Digital Communications and Networks*, vol. 2, no. 1, pp. 24 – 34, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352864815000681

[23] C. Garripoli, M. Mercuri, P. Karsmakers, P. J. Soh, G. Crupi, G. A. E. Vandenbosch, C. Pace, P. Leroux, and D. Schreurs, "Embedded dsp-based telehealth radar system for remote in-door fall detection," *IEEE Journal of Biomedical and Health Informatics*, vol. 19, no. 1, pp. 92–101, Jan 2015.

[24] S. Tomii and T. Ohtsuki, "Falling detection using multiple doppler sensors," in *2012 IEEE 14th International Conference on e-Health Networking, Applications and Services (Healthcom)*, Oct 2012, pp. 196–201.

[25] Y. Wang, K. Wu, and L. M. Ni, "Wifall: Device-free fall detection by wireless networks," *IEEE Transactions on Mobile Computing*, vol. 16, no. 2, pp. 581–594, Feb 2017.

[26] H. Rimminen, J. Lindström, M. Linnavuo, and R. Sepponen, "Detection of falls among the elderly by a floor sensor using the electric near field," *IEEE Transactions on Information Technology in Biomedicine*, vol. 14, no. 6, pp. 1475–1476, Nov 2010.

[27] "Future-shape sensfloor," 2017. [Online]. Available: https://future-shape.com/en/system/

[28] J. Clemente, F. Li, M. Valero, and W. Song, "Smart seismic sensing for indoor fall detection, location and notification," *IEEE Journal of Biomedical and Health Informatics*, pp. 1–1, 2019.

[29] M. Alwan, P. J. Rajendran, S. Kell, D. Mack, S. Dalal, M. Wolfe, and R. Felder, "A smart and passive floor-vibration based fall detector for elderly," in *2006 2nd International Conference on Information Communication Technologies*, vol. 1, April 2006, pp. 1003–1007.

[30] Official Journal of the European Union, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation)," April 2016.

[31] Y. Zou, J. Zhu, X. Wang, and L. Hanzo, "A survey on wireless security: Technical challenges, recent advances, and future trends," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1727–1765, Sep. 2016.

[32] "Mqtt homepage," 2019, visited on: 6/18/2019. [Online]. Available: http://mqtt.org/

[33] "Mosquitto website," 2019, visited on: 6/18/2019. [Online]. Available: https://mosquitto.org/man/mosquitto-conf-5.html

[34] “Mqtt python client,” 2019, visited on: 6/18/2019. [Online]. Available: https://www.eclipse.org/paho/clients/python/docs/

[35] “Twilio voice api,” 2019, visited on: 6/19/2019. [Online]. Available: https://www.twilio.com/voice

[36] “Flask,” 2010, visited on: 6/19/2019. [Online]. Available: http://flask.pocoo.org/docs/1.0/api/#api

[37] “ngrok,” 2019, visited on: 6/19/2019. [Online]. Available: https://ngrok.com/docs

[38] “user question,” 2019, visited on: 6/19/2019. [Online]. Available: https://stackoverflow.com/questions/54237067/how-to-make-tkinter-gui-thread-safe

[39] “user review,” 2018, visited on: 6/19/2019. [Online]. Available: https://www.reddit.com/r/Python/comments/7rp4xj/threading_a_tkinter_gui_is_hell_my_least_favorite/

[40] “Kivy documentation,” 2019, visited on: 6/19/2019. [Online]. Available: https://kivy.org/doc/stable/guide/lang.html