Activity and Fall Detection in the Habitational Environment

Subsystem: Fall detection algorithm

I. Cornelis and S. Falkena

TU Delft

Activity and Fall Detection in the Habitational Environment

Subsystem: Fall detection algorithm

by

I. Cornelis and S. Falkena

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

Student numbers:	I. Cornelis:	4392701
	S. Falkena:	4293681
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
	ir. K. Rassels,	TU Delft, jury member
	Dr. D. Elkouss Coronas,	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 a fall detection algorithm for a fall detection system using a pressure based floor sensor. The goal of the system is to detect falls and alarm the relevant personnel when an elderly person has fallen.

The fall detection algorithm has a strong connection with the interface subsystem, which uses the algorithm as a function. The interface subsystem supplies matrices containing the raw sensor values of the pressure floor.

The algorithm has been divided into multiple sub-algorithms. First, pre-processing: data linearization was applied on the raw sensor values and the sensor matrix was processed such that an image formed that looked like the real world scenario. Second, image processing techniques were applied to detect contours. Contours were being tracked through time, and being grouped. The characteristics of the contours and groups were used to classify falls. Tests have been done to validate the behaviour of the algorithm, from which an average false negative ratio of 30% was achieved in a time window of 30 seconds.

The created prototype proves that image processing is a viable tool for detecting falls with the use of a pressure-based floor sensor.

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 and Acknowledgements

For the duration of 8 weeks, we have been working on designing and testing a system to detect activity and falls. We have done this together with four other group members. It was not only informative but also a very pleasant time where we learned a lot from each other. Therefore we would like to thank our team members: Bram den Ouden, Hendrik Jan Kruijsse, Sander Delfos, and Erik Granneman for this great time.

We would like to thank our project proposer and supervisor Paddy French for the help and freedom within the project. We are very grateful to Kianoush Rassels for supplying us with endless ideas for the project and keeping us critical on our own work. And last, but not least, we would like to thank Gerard Janssen for his support and guidance for the project.

We started the bachelor of Electrical Engineering a couple of years ago, not exactly knowing what was ahead of us. In these years, we have grown as a person, gaining skills, professionalism and knowledge.

With both of us continuing our studies with a Master of Embedded systems, the project was very well suited for both of us, making it no effort to spend many hours behind our laptops writing code and having endless discussions on algorithms.

We would like to thank everyone that made this possible. The TU Delft, teachers, family and friends. Without them, this would not have been possible.

I. Cornelis and S. Falkena
Delft, Friday 21st 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 report focuses on the fall detection algorithm. In the first chapter, the reader is supplied with background information about the project, after which in chapter 2, the global system requirements are discussed. Then in chapter 3, the decision is made to go for a floor based sensor system and the subgroups are being discussed. Chapter 1, 2 and 3 are common for all subsystems. Then in chapter 4, the requirements for the fall detection algorithm are being shown, whereafter in chapter 5, the global outline of the algorithm is explained. In chapter 6, the main part of the report, all steps that have been introduced in chapter 5 are being described in detail. Finally, in chapter 7, the fall detection algorithm is being evaluated and a conclusion together with discussion and recommendations will follow.

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).

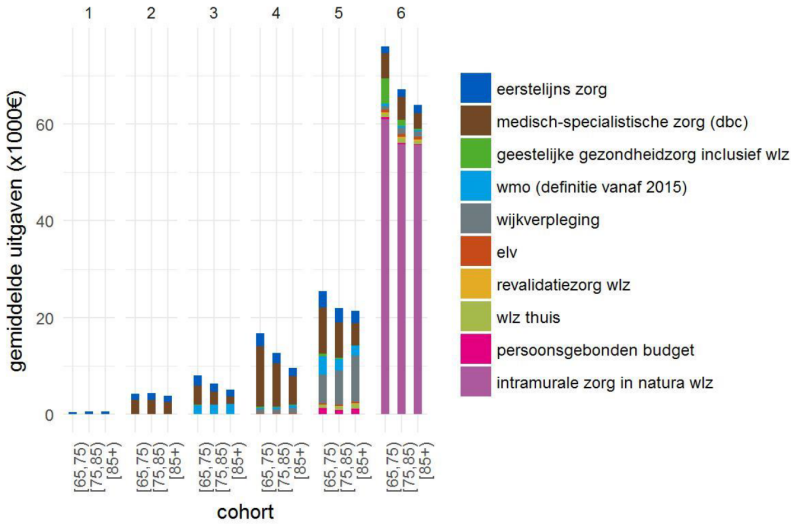


Figure 1.1: Care costs for the Dutch government in 2018

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.

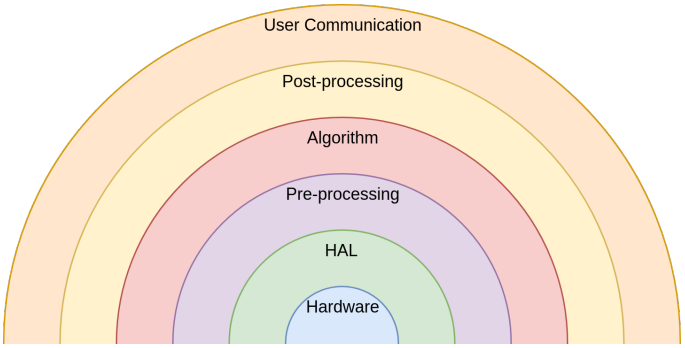


Figure 3.1: System layers

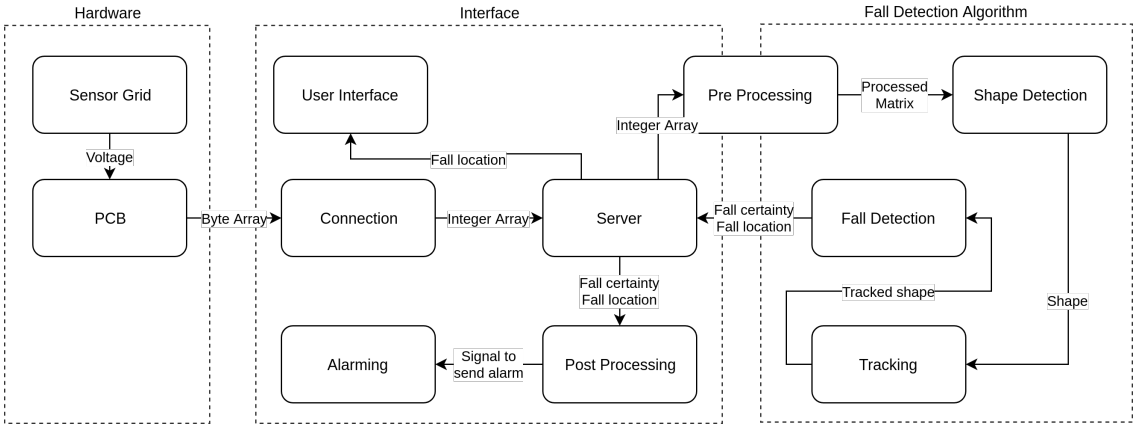


Figure 3.2: Advanced system overview with data flow

4
Fall detection Algorithm

From here on, this report will only focus on the fall detection algorithm part of the fall detection system. The goal of this subsystem can be described in just a few words: design an algorithm that will take data from the sensor array and return whether or not someone has fallen. However, this algorithm is actually quite complex and will need multiple sub-algorithms to come to a correct conclusion.

In this chapter, the subsystem requirements will be derived. These requirements are based on the initial design decision and the global system requirements as described in chapter 2.

4.1. Requirements

Of all the requirements derived in the previous chapter, only five remain relevant for this subsystem. Two functional requirements:

R1.1 Both slow and hard falls should be detected

R1.4 Furniture should not influence the decision making

R1.7 The system should only report falls when one person is present in the room.

And three non-functional:

R2.1 A fall must be detected with false negative rate lower than 10%

R2.2 A fall must be detected with a false positive rate lower than 20%

R2.7 The system should not rely on feedback of the user

Other requirements such as **R2.3** (No usage of cameras) are already covered by the decision to use a floor-mounted sensor. Requirements such as **R1.2** (Signal for alarm within one minute) are among the responsibilities of the interface subsystem.

4.2. Additional subsystem requirements

The fall detection subsystem must be able to connect with the interface subsystem. This leads to additional subsystem requirements. All requirements listed below are still top level for the algorithm. Additional detailed technical requirements will be discussed in the explanation of the sub-algorithms. The false negative and positive rate of requirements **R2.1** and **R2.2** are those of the total system. The algorithm must decide if a person has fallen on each input matrix. The interface subsystem will time average this over 30 seconds and set a threshold of 70% of all falls reported before truly calling for help. [4] This explains the subsystem requirements **SR.6**, **SR.7**

SR.1 The algorithm should be callable as a function.
The algorithm will be used by the interface subsystem and must, therefore, be a function.

SR.2 The algorithm input must consist of a matrix containing the sensor values representing the actual physical grid dimensions.
This is one measurement of the complete floor.

SR.3 The algorithm must return the following values on each input matrix:

(a) **The certainty that a person has fallen.**
This should be scaled from 0 to 1.

(b) **The location of the person**
The location will be used by the interface subsystem for a habit tracker. This should be returned as a coordinate (x,y), where this coordinate represents a location on the floor. There should be one location, independent of how a person is behaving. (standing, walking, laying, jumping)

SR.4 It must be possible to detect a walking movement on the floor
Footsteps give information about the location of a person within the room and the persons' behaviour.

SR.5 The update frequency should have a minimum of 14Hz
The algorithm must be able to process each input matrix at least as fast as they are given so that the algorithm is not the limiting factor for processing time. Since the data is delivered at 14 Hz, the minimum update frequency must be 14 Hz.

SR.6 When a person falls, the false negative ratio must at least be 30%
This false negative ratio must be measured during a time window of 30 seconds.

SR.7 When no person has fallen, the false positive ratio must be below 70%
This false positive ratio must be measured during a time window of 30 seconds.

5
Overview

This chapter covers the overview of the algorithm. A general approach for designing the algorithm will be chosen, after which the steps that make the algorithm will be described on a surface level. Finally, a Programming language will be chosen.

5.1. Approach

Before the algorithm can be designed, a general approach must be chosen. The input has already been determined: sensor values in the input matrix corresponding to the physical location of the sensors. This is similar to an image, where pixels correspond to where light originally hit the sensor. Additionally, the pressure applied to the sensors corresponds to a higher sensor value, in the same way as more light corresponds to a brighter pixel. Figure 5.1 shows the data from the sensor mat as received from the interface subsystem as a grayscale image. Because of this similarity with a low-resolution image, it makes sense to design an algorithm that uses image processing.

An additional option is to use machine learning. This is often used together with camera's to detect human forms. This is because machine learning has improved drastically over the last decade, which has made it easier to experiment with. By recognising human forms, it is possible to detect whether a person is standing upright, kneeling or laying down. Often, supervised machine learning methods like neural networks are used to classify between different ways of falling [12]. The difficulty of machine learning is getting enough real fall data to train the system. This would mean that the test persons would need to fall often, which is undesired and unsafe. Simulating a fall by letting younger people fall in different ways does give more data, but this data is not representative of the falling behaviour of elderly [2].

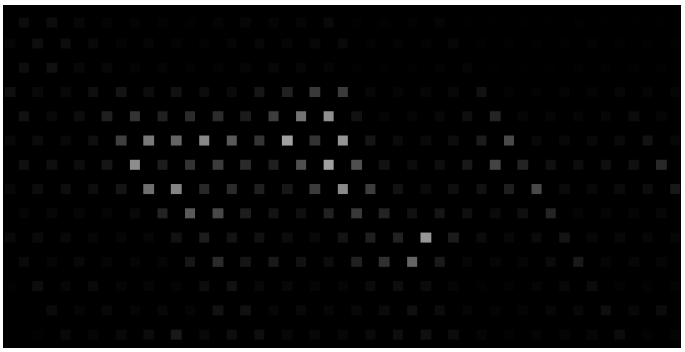


Figure 5.1: Data as received from the interface module. The data represents a person laying on the sensor array with the head to the left.

5.2. Design of the algorithm

In order to draw conclusions on the data shown in figure 5.1, the algorithm must know which 'bright spots' belong together. One solution is to measure contours around these 'bright spots'. However, before these contours can be detected, a couple of pre-processing steps need to be made. These steps will be explained on a surface level in the next sections. See chapter 6 for the detailed description. Figure 5.2 can be used as a guide to get a clearer overview of the algorithm.

5.2.1. Pre-processing

The relation between the applied pressure and the brightness of a pixel is highly non-linear. This causes the sensor array to be more sensitive to low pressure. This relation is not ideal for later processing. Therefore the data is made linear. Next, the background needs to be removed. The background mainly consists of furniture applying pressure to the floor. According to requirement **R1.4**, the influence of furniture should be removed. Finally, the 'bright spots' in the image need to be re-sized. Right now, they represent the actual area of a sensor, which means that these pixels are not connected. These true-data-pixels will be re-sized such that they are connected.

5.2.2. Contour detection and tracking

With the sensor data in a more usable format, the contours can be detected. Features such as the area and the weight of the contours can be calculated. To help classifications, the contours will also be tracked between frames. This allows the algorithm to see changes in the area of the contours. Next, the tracked contours are grouped together to form a larger contour. The larger contour should contain all the contours generated by a person laying on the floor. These larger contours are tracked as well.

5.2.3. Fall classification

Finally, a conclusion on a fall can be made using all the results of the previous steps. This includes the total area of grouped contours, the total weight of grouped contours, the shape of these contours and more. The conclusion is fed back to the interface group.

5.2.4. Person classification

Requirement **SR.4** states that the location of the person must be returned to the interface module. Furthermore, requirement **R1.7** states that the system should not report a fall if more than one person is present in a room. This means that the algorithm must know when more than one person is present. An overview of the total algorithm can be seen in figure 5.2.

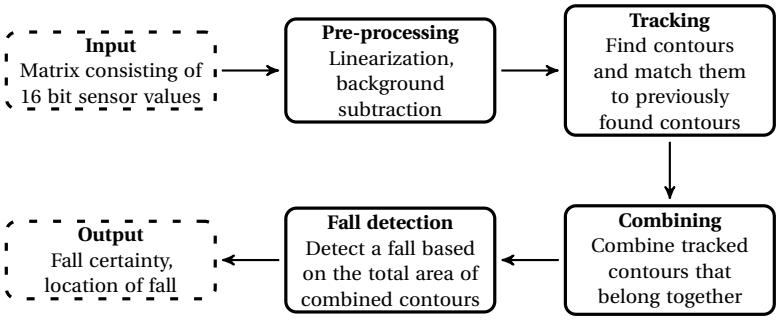


Figure 5.2: Flowchart of the algorithm.

5.3. Choosing a platform

The algorithm will need to be written in some programming language. This needs to be a fast language as image processing is a CPU intensive task. However, it also needs to be easy to prototype as the project needs to be completed in ten weeks, and have sufficient packages to work with image-processing. A good candidate for a fast prototyping language is Python. This is a scripting language with clear syntax. For a scripting language is it relatively fast due to close integration with time-tested and highly optimised codes written in C and Fortran [31]. Python has an open source image processing toolbox called OpenCV-Python. This toolbox is natively written in C++, which makes it faster than other image processing options purely written in Python. OpenCV-Python is the OpenCV API[1] for Python. This allows for writing all code in Python [32]. The final reason to use Python is consistency between subgroups, as the interface subsystem also uses Python. [4]

[1] API: Application programming interface

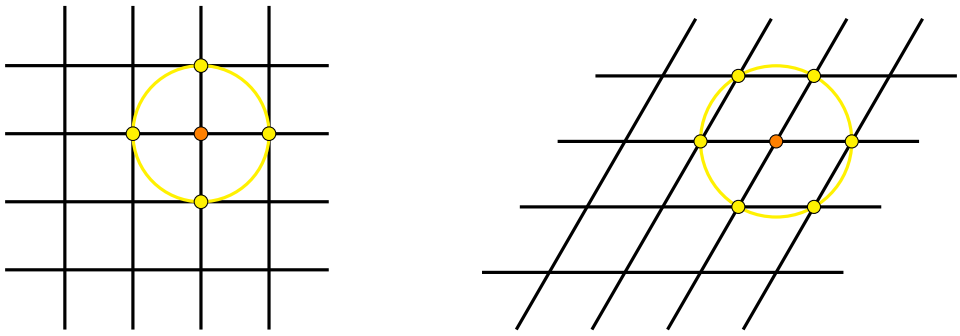
6
Detailed Description

6.1. Input description

Before diving into the details of the algorithm, the input will be described in more detail. The physical sensor array consists of 16x32 or 512 pressure sensors. Each sensor is sampled using a single ten bit ADC. The measured value is then cast to a 16-bit unsigned integer of which only the ten MSBs [1] contain information. All 512 sensors are sampled in roughly 70 ms or at 14 Hz. This would lead one to believe that the input would consist of a 16x32, 16-bit array. However, this is not the case, as will be explained in the next section.

6.1.1. Spacial uniform sampling

Initial testing showed that the sensor array was better at detecting shapes that were aligned with a rectangular sensor grid. Therefore, it has been decided to use a grid in which the sensors are spread more uniformly. An example can be seen in figure 6.1. In the straight-grid style (fig. 6.1a), the orange node has four equally spaced neighbours (in yellow). In the angled-grid style (fig. 6.1b), the same node has six, equally spaced neighbours (in yellow). This causes the shape, visible after placing an object on the sensor array, to be less dependent on the orientation of the object.



(a) straight grid

(b) angled grid

Figure 6.1: Grid styles

This angled grid style cannot be represented using a 16x32 array as the matrix needs be skewed. Thus, the matrix is resized to a larger shape and the true-data-pixels are moved such that they represent the angled grid as shown in figure 6.1b. The new points in between true-data-pixels are left at zero. This leads to the image as can be seen in figure 5.1. The interface module takes care of skewing the matrix as the orientation of multiple sensor arrays needs to be taken into account.[4] Thus, the input for the algorithm consists of a skewed matrix. The size of this skewed matrix depends on the number of sensor arrays that are used and is therefore variable.

[1] MSB: most significant bit

6.1.2. Name convention

With the decision to use image processing, it makes sense to use names more common for images. For better readability, the name conventions as used throughout this document are summed up:

- **Sensor array**: the physical 16x32 sensor array.
- **Sensor**: a physical sensor as present on array.
- **Sensor Value**: the value of a sensor as captured by the ten-bit ADC.
- **Frame**: the skewed input matrix containing all sensor values.
- **Pixel**: a single entry in a frame
- **Pixel value**: also intensity or brightness, the digital value of a single pixel.

6.1.3. Pressure curve

The relation between pressure applied to a sensor and the brightness of a pixel is highly non-linear. This relation can be seen in figure 6.2. As said before, the data is captured using a 10 bit ADC. Thus, the maximum value measurable would be 1023. This is however not the case, as the curve saturates at an intensity of around 700. This is due to voltage drops in the measurement setup.[3] It is also clear that the sensor is far more sensitive for low pressure, and that it only offers its best resolution in the range from zero to about 30 N/cm^2. During foot pressure measurements, Birtane et. al. found that feet, with an average surface area of around 81.2cm^2, applied a maximum pressure of around 40 N/cm^2 during normal walking. [33]. Standing resulted in a lower pressure in the range of 4 $N/cm2$. Measurements from another research by Pu et. al can be converted using the average foot surface area of 81.2cm^2 so the results can be compared. Pu et. al found a maximum pressure of around 10 - 14 N/cm^2 for walking, running and small jumps [34]. This implies that the range of linearization is most interesting in the range of 0 to 40 N/cm^2.

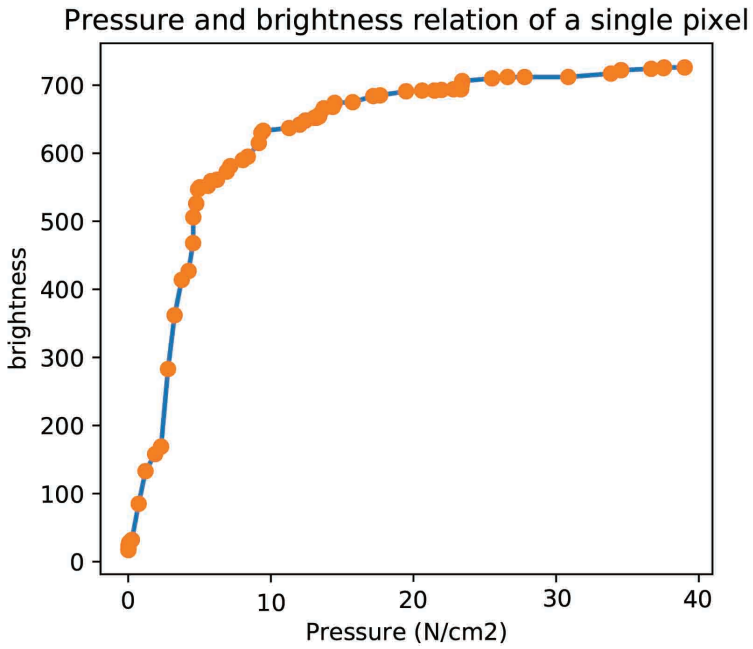


Figure 6.2: The relation between pressure on a single sensor and the brightness of a pixel.

6.2. Pre-processing

This section covers the steps that are necessary before contours can be detected.

6.2.1. Linearization

As stated in section 6.1.3, the pressure vs. brightness curve of the sensor is highly non-linear. To make the behaviour match closer to reality, a linear reaction to pressure would be ideal. This can be achieved by fitting the measured data from figure 6.2 and then converting it. Doing the same experiment as shown in figure 6.2, on a smaller pressure range with higher accuracy leads to figure 6.3. It can be seen from this figure,that the region of interest is from 0 to 27 N/cm^2. The part that is interesting for the algorithm lays between 0 to about 30 N/cm^2. This means that brightness values above 730 are not relevant. As the asymptote of the curve lays around 750, the range of 730 to 750 is difficult to measure correctly as the curve is barely changing, making it difficult to safely make any conclusions above 27 N/cm^2. The linearization function should take care of two things:

1. Remapping of values from 10 bit (from the ADC) to 8 bit, as most OpenCV function work in 8 bit data.[2]
2. convert the data such that the pressure-brightness curve is linear and passes through the point $(0,0)$, when no pressure is applied, and the point $(27,255)$, when the highest relevant pressure is applied.

The curve from figure 6.2 can be fitted with an equation of the form:

$$y_1 = c - \frac{a}{(x+b)^2} \tag{6.1}$$

where a, b and c are constants and a is the gain, b is the shift in the x-direction, and c is the horizontal asymptote. Using a fit tool, it turns out that $a = 235885.14$, $b = 16.45$, $c = 879.09$ which can be seen by the blue curve in figure 6.3.

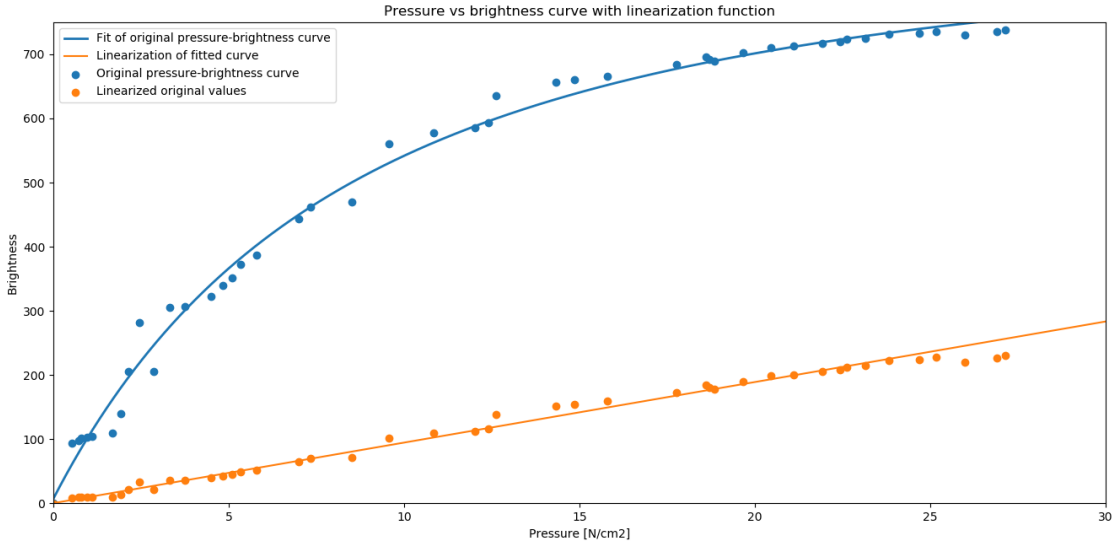


Figure 6.3: Measured pressure-brightness curve with fit and linearized data

The non-linear curve needs to be converted to a linear equation with the form:

$$y_2 = \frac{a}{b} * x \tag{6.2}$$

[2] 16-bit and 32-bit images are also possible but are inconstantly supported by OpenCV

where a and b determine the slope of the function. As the linear curve is supposed to go to the point (0,0) and the point (27, 255), equation 6.1 and 6.2 combine to the linearization formula, mapping y_1 to y_2:

$$y_2 = \frac{255}{27} * \sqrt{\frac{235885.14}{879.09 - y_1} - 16.45} \tag{6.3}$$

Validation and discussion
After the linearization formula was implemented, it was tested on the sensor array. A force was applied to the sensor array with a scale underneath it to measure weight. Having a surface of $0.5x0.5mm$, the weight could be converted to pressure in N/cm^2. In figure 6.4 it can be seen that the acquired data fits the ideal curve best for lower pressures. The segment from 15 to 25 N/cm^2 is less accurate, possibly due to a measuring error, as the force was applied by a person. Furthermore, the linearization function is more sensitive to small errors in this range. However, it is clear that linearized data passes through the required points (0,0), and (27,255). Additionally, it is possible to assign a unit to brightness: $27/255 \approx 0.11N \cdot cm^{-2} \cdot bin^{-1}$.

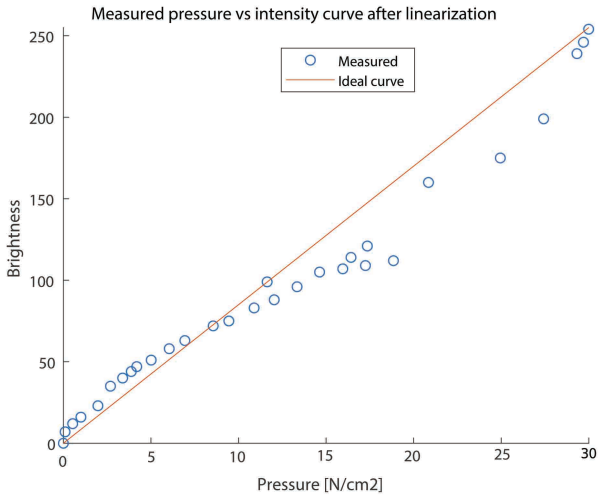


Figure 6.4: Measured pressure-brightness linearized data

The conversion from ten to eight bit does lead to a loss of resolution. However, not all of the possible resolution is used to begin with, as only zero to about 750 can be used. It is entirely possible to convert the algorithm to use 16-bit frame representation and convert back and forth for OpenCV functions that do not support 16-bit and above. An example of such a function is the `findContours()` function that will be described in section 6.3.1. However, the eight-bit representation offers more than enough resolution for contour detection.

6.2.2. Background subtraction

Requirement **R1.4** states that furniture or other static objects should not influence decision making. This means that, due to how the fall detection algorithm is structured, static objects should not be detectable as contours and need to be filtered out. Two requirements have been set for designing a background subtract function:

1. The function should take an argument that defines how long an object must be static before becoming background. This is to avoid classifying a motionless person on the floor as background. This value will be at least 30 minutes.

2. The function needs the option to reset during run time, which should immediately make everything present at that time background.

Three techniques have been considered and tested against these requirements: Built-in OpenCV background subtraction algorithms, high-pass filtering pixel values in time, and thresholding.

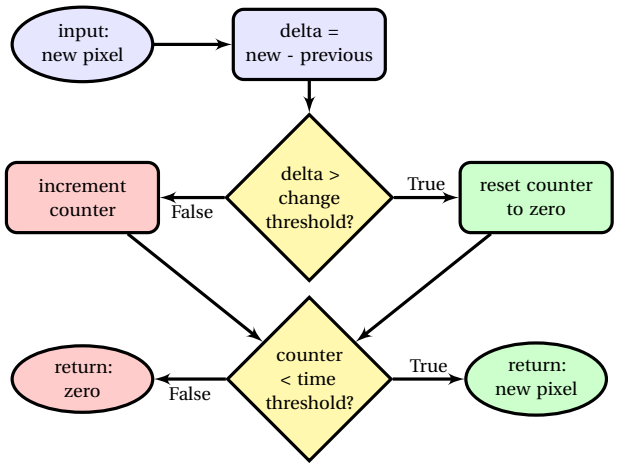


Figure 6.5: Flowchart of the background subtract algorithm

Built-in OpenCV functions
OpenCV comes with a couple of built-in background subtraction methods, focused on working with video material of in- and outdoor situations with low light change. It uses pixel statistics and frame differencing, making it very fast for video feed [35]. This makes it difficult to use for background subtraction for the sensor array, as the data is inherently different. Also, a pixel's behaviour is dependent on its surrounding pixels, making the background "fade out" of the picture, which would still lead to falsely detected contours.

Digital high-pass filtering
High-pass filtering has been tested as well. The value of a pixel in a series of frames can be put through a digital high pass-filter. Both FIR[3] and IIR [4] filters have been tested. However, meeting the second requirement would mean designing a filter with a very low cutoff frequency. This leads to the number of filter coefficients for the FIR filter to be in excess of a thousand. Each coefficient would need its own frame buffer, which all need to be present in the memory. Therefore, this option has been dropped. IIR filters were more promising, only needing a couple of frame buffers. However, the IIR filter had three problems. First, stability and resolution. The low cutoff frequency leads to very sensitive filter coefficients, with far more resolution than the input. The input and output of the filters are pixels, which can only have integer values from zero to 255. This causes the filters to not perform as designed and sometimes to be unstable. Secondly, these filters work gradually, thus influencing relevant data before fully classifying it as background. Finally, it is difficult to implement a reset that makes everything background, as all frame buffers need to be set such that they result in an output of zero. For these reasons, IIR filters have also been dropped.

Thresholding
Finally, an algorithm has been designed that counts how long a pixel does not change in value. If the time threshold of requirement 1 has been reached, it sets the pixel value to zero. If a change is detected, the pixel value is allowed to pass through again. A flow chart of this algorithm can be seen in figure 6.5. During initialisation, the previous pixel value and the counter are set to zero. If a reset is required, the counter is set to a value greater than the time threshold, thus the algorithm will return zero. This algorithm meets the two requirements for the background subtract function and has been tested to work according to the input parameters.

[3] FIR: Finite impulse response
[4] IIR: Infinite impulse response

Discussion
One of the advantages of the thresholding algorithm is the binary classification of background. This means that data is allowed to pass through unaltered until the time threshold of requirement 1 is reached. This is preferred over other implementations that slowly make objects background, until they are fully removed. However, the algorithm can be improved. for example, if some region in a frame is classified as background, it only takes one frame of change to lift this classification. It would be better if the algorithm remembers that the region was classified as background before the change, and reclassifies it if no further change is detected. Furthermore, with the linearized data, it should be possible to subtract constant background from all future inputs. This would lead to more accurate measurements of dynamic inputs.

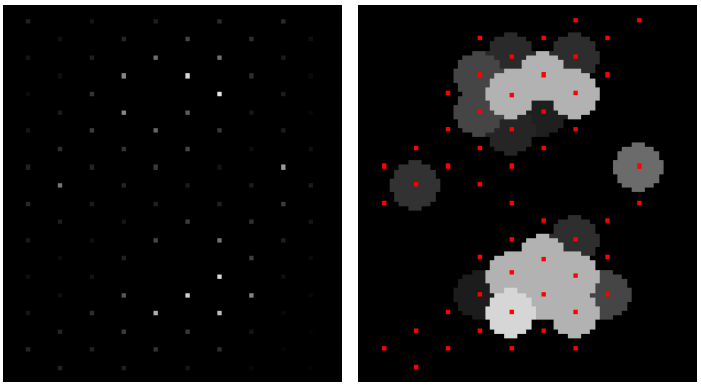
6.2.3. Pixel resize

At this point, the pixel values have only been altered by the linearization and background subtraction. The frame still has the form as shown in figure 6.6a. As this form does not look very natural and it is not yet ready for contour detection. Thus, data will be processed further.

The pixel resize step comes with the following requirements:

1. Raw pixel data should be scaled up, so that a closed contour can be drawn around shapes.
2. Scaling up should be done using a circular form so that a pixel value influences all of its adjacent true data pixels.
3. All pixels in between true data pixels should be linearly interpolated between the values of these true data pixels.

Resizing pixels will be done using OpenCV's dilate function. This function enlarges any pixel that has a nonzero value. It is possible to design custom kernel functions that it will use to enlarge the pixel. As a circular form is desired (as explained with figure 6.1b), a circular kernel function has been made. The dilation step can be seen in figure 6.6b, where for convenience the true data pixels that are nonzero are highlighted in red. Here, it can be seen that the dilation size is chosen to be equal to the distance between true data pixels.



(a) The raw image (b) Dilated pixels

Figure 6.6: The different stages in pixel resizing

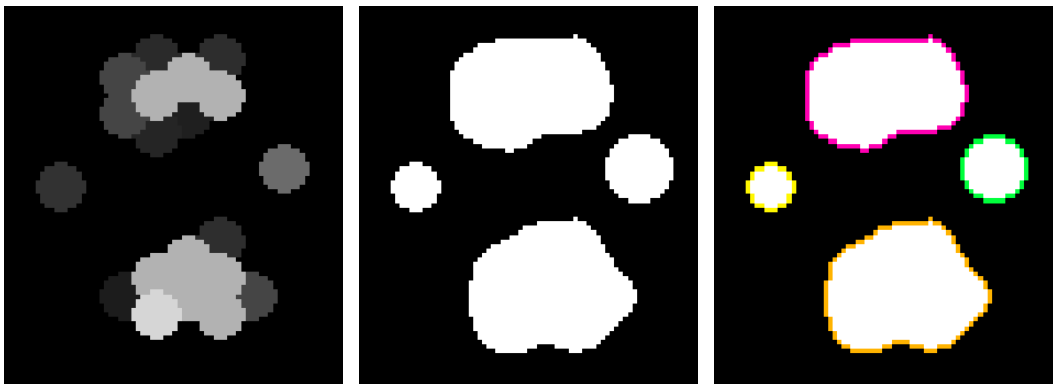
Discussion
The solution to dilate only covers two of the three requirements. Furthermore, bright pixels are laid on top of less bright pixels, this likely means an over-estimation of the real pressure applied. Not linearizing the pixels between two true data pixels, adds to this over-estimation of applied pressure. An efficient way to implement this has not yet been found. It has been tried to use blur to approximate this, but this influenced the linearization as discussed in section 6.2.1, and has therefore been dropped. Finally, it is not yet clear if the third requirement is even the right approach. It quite likely that in reality, the pressure curve between two points is not linear at all.

6.3. Contours

As stated before, contours will be used for fall classification. Pre-processing has made the data ready for contour detection. At this point in the algorithm, the data looks like figure 6.6b.

6.3.1. Contour detection

Contours need to be detected in each frame. In OpenCV, two steps are needed to detect contours in an image.[35] First, a frame is converted to a binary image based on a threshold. This means that pixels with a value below or equal the threshold are set to zero, and pixels with a value above the threshold are set to 255, the maximum value a pixel can take. An example of such an operation can be seen in figure 6.7, where a base frame (fig. 6.7a) has been converted to a binary image (fig. 6.7b). Next, the OpenCV function `findContours()` is used to find transitions between black and white pixels and connect these to form closed contours. It returns arrays of (x, y) locations that define each contour. The result can be seen in figure 6.7c, where each colour represents a different contour.



(a) The base image (b) Conversion to a binary image (c) The detected contours

Figure 6.7: The different stages in contour detection

6.3.2. Contour Tracking

Contours are detected on each frame. However, the algorithm does not yet know that contours between frames are related. The contours need to be tracked to be able to tell something about contour parameters over time. The algorithm that will take care of this has a few requirements:

1. The tracking algorithm must be able to track multiple contours.
2. The tracking algorithm must be able to track contours, even if they disappear for a couple of frames.

A very simple algorithm has been written that tracks contours based on distance. To calculate the distance between contours, contour centroids will be used. Calculating a centroid requires two steps. First, the contour moments are needed, which are defined by [35, p. 429]:

$$M_{p,q} = \sum_{i=1}^{N} I(x_i, y_i) x_i^p y_i^p \tag{6.4}$$

In this equation, the moment $m_{p,q}$ is defined as the sum over all pixel values in a frame that are part of a contour, where the contour consists of N points. A pixel value at location (x, y) is multiplied by a factor x^p, y^q. If a moment is calculated on a binary image such displayed in figure 6.7b, then, equation 6.4 simplifies to eq. 6.5.

$$M_{p,q} = \sum_{i=1}^{N} x_i^p y_i^p \tag{6.5}$$

If the moment m_{00} is calculated, just the length of the contour comes out. However, with the combination of M_{10} and M_{01}, the contour centroid can be calculated by using equation 6.6, which really just the takes the average x and y location of a contour. [35, p. 430]

$$C = \left(\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}} \right) \tag{6.6}$$

In this equation, C is the contour centroid consisting of a (x, y) coordinate. The centroids of contours detected in a new frame are compared to the centroids of previously found contours by constructing a distance matrix which is defined by equation 6.7. In this equation, $C_{new,i}$ is the i^{th} centroid of the newly found contours, and $C_{known,j}$ is the j^{th} centroid of the already tracked contours.

$$D_{i,j} = \left|\left| C_{new,i} - C_{known,j} \right|\right| \tag{6.7}$$

The tracking algorithm takes this distance matrix and looks for the smallest value. If, say, $D_{1,2}$ contains the smallest value, then the first newly found contour is matched to the second already tracked contour. The entry at $D_{1,2}$ is replaced with a floating point infinite value. With this new distance matrix, the algorithm looks for the next smallest value. Setting the entry to infinite makes sure that the next smallest entry can be found. This process continues until all new contours are matched to an already tracked contour.

In the case that the number of new contours exceeds the number of already tracked contours, the unmatched new contours will be registered as tracked contours. In the other case, when the number of tracked contours exceeds the number of new contours, a counter, unique to all tracked contours, is updated. If this counter reaches a certain maximum value, the tracked contour is unregistered.

An overview of the total tracking algorithm can be seen in figure 6.8. In this figure, `min_index(D)` represents a function that returns the i, j index that contains the smallest entry in the distance matrix. N_{loop} is a variable that represents the n^{th} time the loop has been run, N_n is the number of new contours, N_k is the number of tracked (known) contours. Finally, `threshold` is a variable that determines how long a tracked contour can remain disappeared before it is deleted.

The algorithm as implemented in python contains some additional steps that are not shown in the algorithm flow chart. For instance, when a tracked contour is matched, its 'disappeared counter' is reset to zero. Furthermore, the algorithm makes sure that it does not match contours that are too far apart. This is implemented by breaking the loop when the result of `min_index(D)` is larger than a certain distance threshold.

Tracking discussion
The algorithm works as expected, and correctly tracks multiple contours. However, when two contours move past each other, they temporally become a single contour. When these contours split again, it is a complete chance if they are matched as before the meet-up. This could be improved by taking the movement of the contours in to account as well.

6.3.3. Contour grouping

When a person falls, multiple contours are detected. The algorithm needs to know that these contours belong together. This is done by grouping contours based on distance. Consider three contours: `A, B` and `C`, a maximum distance threshold `d_max` and a function `dist()` that calculates the distance between contours. `A, B` and `C` should be grouped according to the following rules:

1. if `dist(A, B) < d_max`, then `A` belongs to the same group as `B`, else not.
2. if the following conditions hold, then `C` belongs to the same group as `A` and `B`:
   (a) `dist(A, B) < d_max`
   (b) `dist(B, C) < d_max`
   (c) `dist(A, C) > d_max`

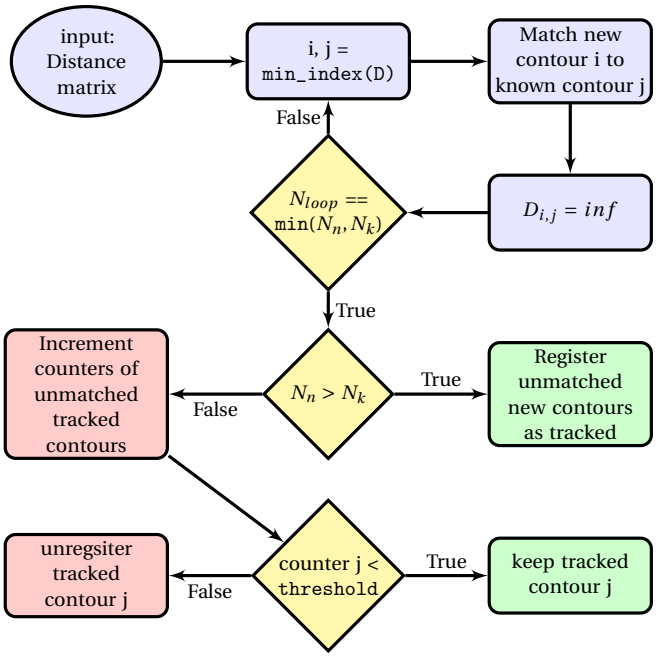
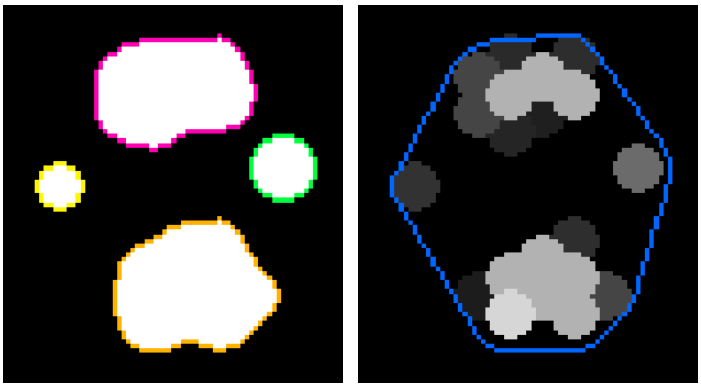


Figure 6.8: Flowchart of the tracking algorithm

The grouping algorithm first constructs a sub-group for each contour according to rule 1. Then, it checks if these sub-groups overlap. If they do, the two sub-groups are added to each other. This continues until no overlap is found. The result is one or more groups that follow both rule 1 and rule 2.

The contours in a group are added to each other and converted to a convex shape. This means that the internal angle between any three sequential points in the contour is less than 180 degrees. The result of this algorithm can be seen in figure 6.9, where the contours in fig. 6.9a are grouped and converted to a convex shape, as can be seen in fig. 6.9b. The contours are also tracked using the same tracker as for contours.



(a) The detected contours (b) Convex group

Figure 6.9: Contour grouping

Grouping discussion

The grouping algorithm has the similar issues to the tracking algorithm, as both are only based on distance. It could be improved by also taking features such as movement of individual contours in to account.

6.4. Fall classification

To classify a fall, it is important that requirement **R1.1** is satisfied. Different types of falls have different consequences for the algorithm. Therefore multiple contour and group features will be used to test if a person has fallen. These features include:

1. **Group area**: The area of the group in squire centimetres. See appendix A
2. **Group length**: The longest side of a bounding box around a group in centimetres. See appendix B
3. **Contour length**: The longest side of a bounding box around a contour in centimetres. See appendix B
4. **Number of contours**: The number of contours that make up a group. See appendix C
5. **Group Aspect ratio**: The ratio of the width and height of a bounding box around a contour. See appendix D
6. **Group weight**: The sum of pixel values within a group. This value is then multiplied by the constant found in section 6.1.3 to get back to N/cm^2. The result is divided by the gravitational constant to convert to to kg. See appendix E
7. **Group average pressure**: The sum of pixel values within a group, divided by the area. The result is then multiplied by the constant found in section 6.1.3 to get back to N/cm^2. See appendix F
8. **Group maximum pressure**: The maximum pixel value in a group, multiplied by the constant found in section 6.1.3 to get back to N/cm^2. See appendix G

These features will be calculated based on the largest group by area. If they rise above or below a certain threshold, certainty of a fall classification must go up. To determine the correct thresholds for these values, a couple of tests have been designed:

1. **Walk test**: a person walking around on the sensor array.
2. **Chair test**: a person sitting down on a chair and then standing up again.
3. **Fall test**: a person falling down on the sensor array and then taking several positions.

The features as described earlier can be recorded during these tests. The results of these measurements can all be seen in the appendix. The following sections will cover each feature for all test, and determine a suitable threshold. This is an optimisation problem, as in most cases, increasing a threshold has opposite effects on false positive and false negative ratios.

6.4.1. Area

Most falls, independent of their nature, have in common that the person ends up laying on the ground. Therefore, the most important aspect of the fall detection algorithm is detecting a person laying on the ground. This automatically implies that the size and form of the area occupied become important factors for classification. From figures A.1, A.2 and A.3, it can be concluded that the area of a contour group during fall increases beyond $6000cm^2$ with respect to a maximum of around $2000cm^2$ when walking and around $3000cm^2$ when sitting on a chair. This is a relative increase of 100%, making it a suitable parameter for classification.

6.4.2. Length

Similar to area, length is is a feature that is likely to increase when a person falls. The length of a group and contour can be seen in figures B.1 until B.6. These figures show that group length is indeed a suitable parameter for fall detection, as a difference of up to $40cm$ can be seen in the case of a fall. Furthermore, an increase of contour length of a fall is found to be around 10 to $20cm$ with respect to walking and sitting. This makes contour length also a suitable factor for classification.

6.4.3. Number of contours in group

The number of contours that build up a group are plotted versus time in figures C.1, C.2 and C.3. It can be concluded that during a fall the amount of contours increases. This is caused by the different parts of the body touching the floor. Although the increase is relatively much, the absolute increase is less significant. Nevertheless, the increase in contour count is a feature that is present for longer times, so it can be concluded that it is a suitable parameter for classifying falls.

6.4.4. Aspect ratio

The aspect ratio of a group is defined as the length of a contour group divided by its width. The measurements for the aspect ratio are shown in figures D.1, D.2 and D.3. It was expected that a clear value would be found for a person laying on the floor, but it is very difficult to find any characteristic values for walking, sitting on a chair or falling. Therefore, it is decided that the aspect ratio of a group of contours is not a suitable parameter for classification.

6.4.5. Weight

The results of the walk, chair and fall tests for weight are visible in figures E.1, E.2 and E.3. Weight should be relatively constant between tests, as the true weight of the person does not change. It should also be below about 90 kg. Both are however untrue for the measured weight. This is likely due to the over-estimation of total pressure applied in the dilation step in section 6.2.3. This is made believable by the fact that the weight of a person is larger during a fall, when the pressure is divided over a larger area. Although the weight is over-estimated, it can still be used as an indicator for a fall.

6.4.6. Average pressure

The results of the walk, chair and fall tests for average pressure are visible in figures F.1, F.2 and F.3. During the walk test, pressure varies between a value of one and eight, although the average seems to be around four. During the chair test, the average is similar, but the range about three to eight. The fall test is clearer. A large peak is visible at the start if the fall, which is likely related to the test person hitting the floor. This peak is followed by an average pressure that is lower than the measurements of walking and the chair. It should be possible to check if a peak is followed by a period of low pressure.

6.4.7. Maximum pressure

For hard falls, the most important aspect of the fall is that the person has a high impact on the floor. These impacts should be visible as short peaks in the pressure graphs. As the developed system is sampling the mat with 14 Hz, impacts are almost not measurable, as the sampling time is too short. Research by the University of West Bohemia shows that high acceleration forces (13g) are measured from a fall with $30 km/h$ [36]. This all happens within the time window of 100ms which is outside the measuring range of the system. Nevertheless, it is possible to measure changes in pressure applied to the mat. Any movement on the mat that is a bit slower than the short impact of a fall can therefore be distinguished, like for example, falling on the knees. From figure G.1, F.2 and G.3, it can be seen that the maximum pressure for all three situations is quite similar, as expected, because of the difficulty in measuring. Additionally, for a fall it can be seen that there are short, rapid increases in maximum pressure. Therefore, perhaps a derivative function can be used. To conclude, a maximum pressure could be used as a measure for detecting high impact falls but this would need more advanced processing steps in order to make it work.

6.4.8. Resulting thresholds

Everything discussed above leads to the following selection parameters for the fall classification:

1. When the area of a contour group is greater than $3000 cm^2$, the certainty should be increased.
2. When the length of a single contour is greater than $30 cm$, the certainty should be increased.
3. When the length of a contour group is greater than $100 cm$, the certainty should be increased.
4. when the number of contours in a group is than 3, the certainty should be increased

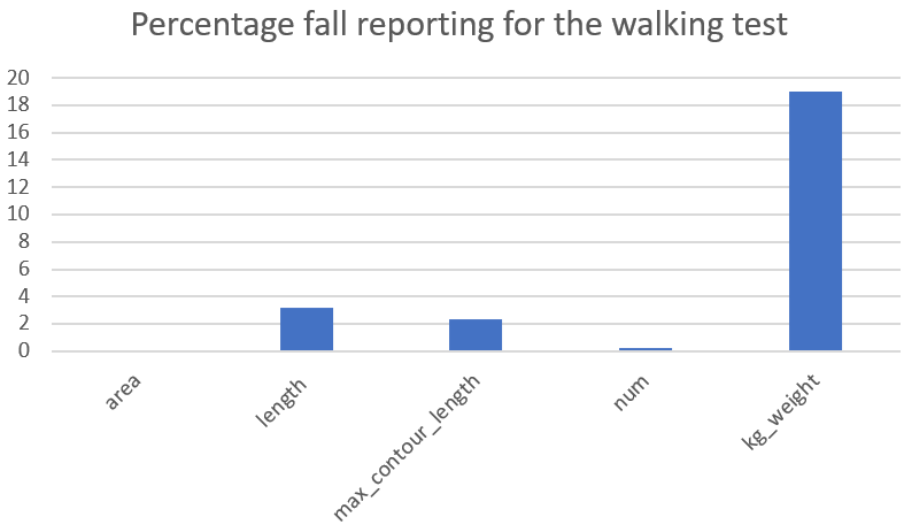


Figure 6.10: Relative times a certain condition increased the fall certainty during the walk test. In this figure, `max_contour_length` is length of the longest contour, `num` is the number of contours in a group, and `kg_weight` is weight of a group.

5. Aspect ratio will not be used.
6. When the weight of a contour group is greater than $200kg$
7. When a peak larger than $8N/cm^2$ is detected, the certainty must be increased as long as the pressure remains between 1 and $4N/cm^2$. At the time of writing, this is not yet implemented.
8. Maximum pressure will not be used.

The algorithm that calculates the certainty of a fall is very simple: it checks the result of each threshold condition on each frame and takes the average of all the results. If all conditions are triggered, 'one' is returned, if none are triggered, 'zero' is returned. Since the number of conditions is five, the certainty resolution is about 20 percent.

6.4.9. Validation

The three tests, the fall, the chair and the walk test have been used to validate the thresholds that were set in the previous section. During each test, the number of times each threshold condition increases the fall certainty, have been recorded. These numbers will be shown as a percentage of total frames.

Walk test

This test should show a low number of threshold condition triggers. This is confirmed by figure 6.10. Here, the weight is the only parameter that raises above 4%. This parameter resulted in increasing the fall certainty incorrectly on 19% of the input frames. In total, the average fall certainty during this test was 5%.

Chair test

The chair test should be more difficult for the algorithm to handle, as the use of a chair results in more detected contours. The results from this test can be seen in figure 6.11. The number of 'threshold condition triggers' was significantly higher than during the walk test. All parameters raise above 12% triggers, except for (group) length. One possibility to reduce the average certainty is to weight this parameter more during the calculation of fall certainty. The average certainly during this test was 13%.

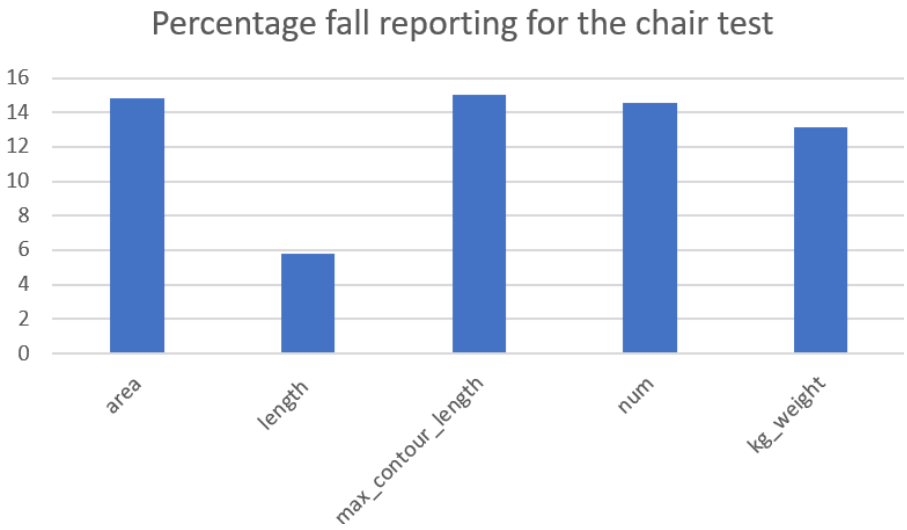


Figure 6.11: Relative times a certain condition increased the fall certainty during the chair test. In this figure, `max_contour_length` is length of the longest contour, `num` is the number of contours in a group, and `kg_weight` is weight of a group.

Fall test

The most important test is the fall test, the results of which can be seen in figure 6.12. As expected, the number of 'threshold condition triggers' is far higher than during the other two tests. The smallest difference between these tests is the area of the chair test and the area of the fall test. The difference between these is still 46%. The average certainty during the fall test was 70%.

6.4.10. Discussion

The method of simple thresholding could definitely be improved. For example, a time aspect for pressure could lead to better fall classification. Furthermore, the way the thresholds were derived was by looking at the plots. Numerical analysis of the gathered data should result in better thresholds. The tests that were used to generate the data were limited, and do not represent all types of falls or all types of daily use. They are however sufficient to show that a fall can be detected on thresholds alone.

6.5. Person classification

Two steps are still remaining: person localisation and multiple person detection. Location is quite simple, as fall is classified on the largest group. Therefore, the centroid of the largest group will be taken as the location of the person. Detecting that multiple people are present is more difficult. The following algorithm has been designed:

1. Check if a group has moved a sufficient amount of distance. This is to make sure that a bag will not trigger multiple person detection.
2. Check if a group has a weight above a certain threshold. This is to make sure that a pet will not trigger multiple person detection.
3. If more than one group satisfies the first two checks, more than one person must be present.

This algorithm has not been implemented and can therefore not yet be tested.

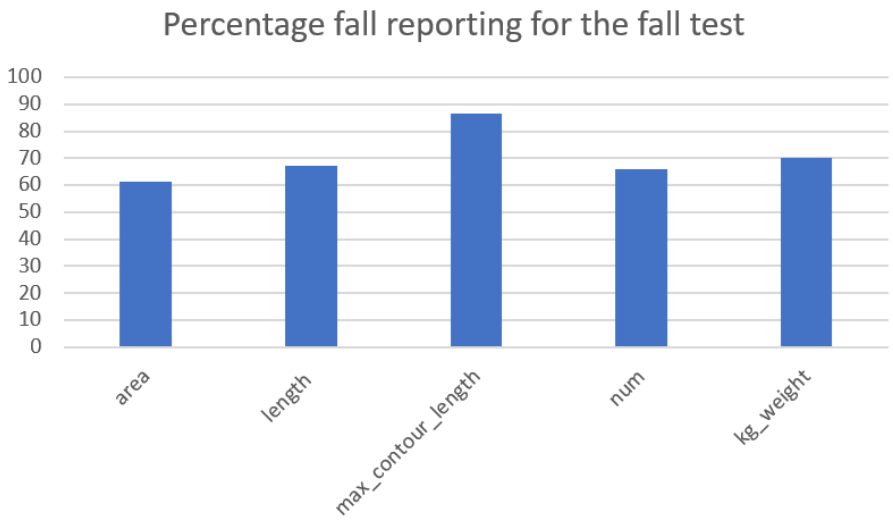


Figure 6.12: Relative times a certain condition increased the fall certainty during the fall test. In this figure, `max_contour_length` is length of the longest contour, `num` is the number of contours in a group, and `kg_weight` is weight of a group.

Further, a premature version of footstep tracking has been designed, in order to detect whether a person might have tripped. This works in the following way:

1. The first footstep is determined. This is the first contour that classifies as a footstep: The aspect ratio is larger than 0.65 and the area is larger than $50cm^2$ and smaller than $120cm^2$.
2. From the first footstep, any other contour in range of 100cm that also would classify as a footstep is nominated to be the next footstep.
3. If the angle of the next footstep is within 90 degrees of the previous footstep the footstep is classified as the next footstep and the algorithm restarts.

6.6. Update rate and parallelization

Requirement **SR.5** states that the update frequency of the system should be 14 Hz. During initial testing, this goal was not reached. The solution came in the form of multiprocessing. The fall detection algorithm runs in a different process than the interface module. The data exchange has been made such, that when the algorithm is given a new frame to process, it immediately returns the result of the previous input frame. This ensures that calls to the fall detection algorithm are non-blocking. With this improvement, the update rate was 14 Hz.

7 Conclusion and Discussion

7.1. System evaluation

In section 4, a total of 11 requirements were set. This section will state for every requirement if it is met. Requirement **R1.1** states that both slow and hard falls should be detected. The system cannot detect impact of hard falls, but as long as a fall ends up with a person laying on the ground, the requirement is met.

The background subtraction algorithm makes sure that furniture does not influence the fall detection algorithm. Therefore, requirement **R1.4** is met. Requirement **R1.7** states that the system should not report falls when more than one person is present in the room. Unfortunately, it was not possible to distinguish multiple people, failing to fulfil this requirement. The false negative ratio of 30%, and the false positive ratio of 70% , as dictated by requirements **SR.6** and **SR.7**, have been reached. The fall report ratio during the fall test was exactly 70%. Requirement **R2.7** states that the system should not rely on feedback of the user. This requirement is met by design.

The algorithm is called as a function from the interface subsystem, from where it receives a matrix containing sensor values. It returns the certainty of a fall and the location of a person. With this, requirements **SR.1** - **SR.3** are met. Requirement **SR.4** states that a walking movement could be detected. Although this feature was prematurely developed, the focus was not put on this detection mechanism, making this requirement not fulfilled. Requirement **SR.5** states that the update frequency should at least be 14Hz and that the algorithm can not be limiting. As the update frequency is measured to be 14Hz which means this requirement is met.

7.2. Conclusion

This report described the fall detection algorithm of a fall and activity detection floor system. The algorithm works by first preparing the incoming data by linearizing it and then uses image processing techniques to detect contours on the floor and group them. Then, different classification techniques (area, length, pressure and number of contours) are being used to classify a fall. Additionally, a walking detection algorithm, a data recorder and simulator have been designed, which can be used for debugging purposes. From the system evaluation, it can be concluded that 9 out of 11 requirements are met. Most importantly, the fall detection algorithm has a 70% success rate for the simple fall test that has been executed.

7.3. Future work and recommendations

The following points are recommended as a good starting point for future work:

- First, including the time aspect in the fall detection algorithm, would be a good addition. Currently, the detection algorithm works frame by frame, but features like an averaging window or a time derivative on the classification features would greatly enhance the detection.
- As described, the system currently does not detect multiple persons. As the algorithm uses the area for classification, this can induce false positives. When multiple persons can be detected, this should also be displayed in the system UI.
- Better and more specific tests are still needed for the algorithm. This is hard, as it is difficult to simulate a realistic fall. Dummy humans can be used to simulate falls against or on objects.
- Automatic classification thresholds can make the system far more robust and applicable to multiple floor types and users.

8 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.

A

Area threshold measurements

During the tests described in section 6.4, the area in square centimetres of the largest group was measured. The results can be seen here.

A.1. Area measurement of the largest group during walking

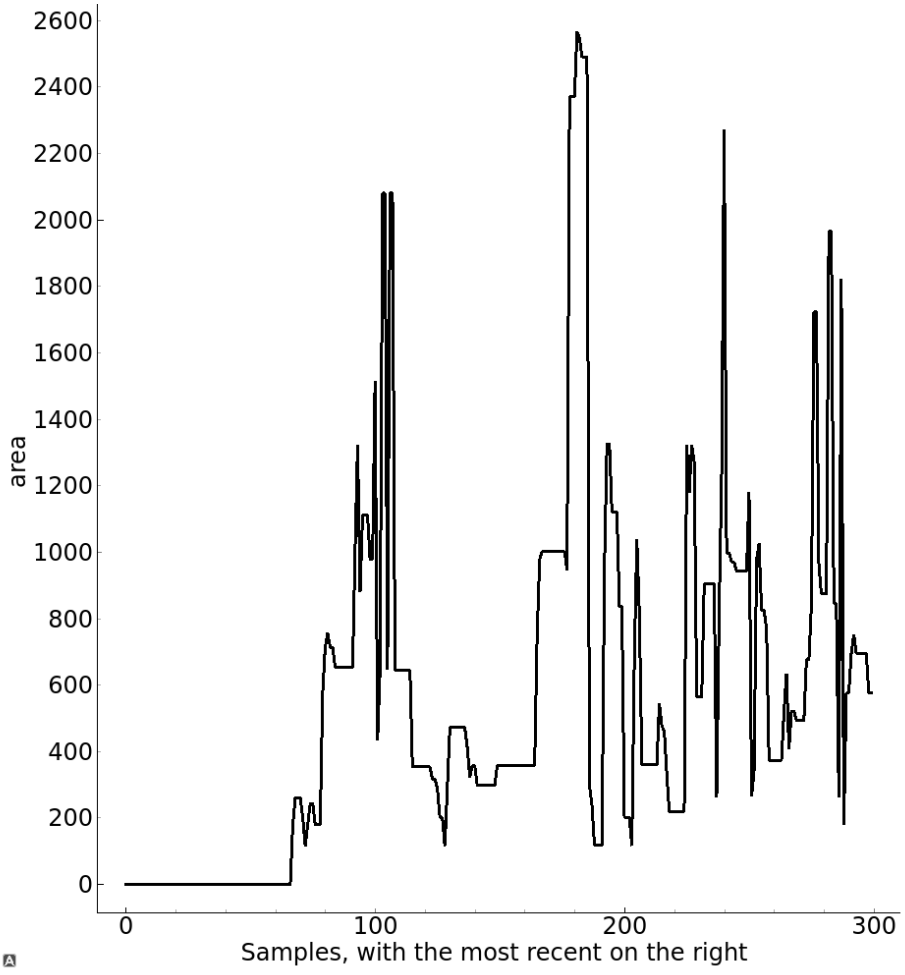


Figure A.1: Area of the largest group in squared centimetres over time. Data was captured during normal walking.

A.2. Area measurement of the largest group while sitting on a chair

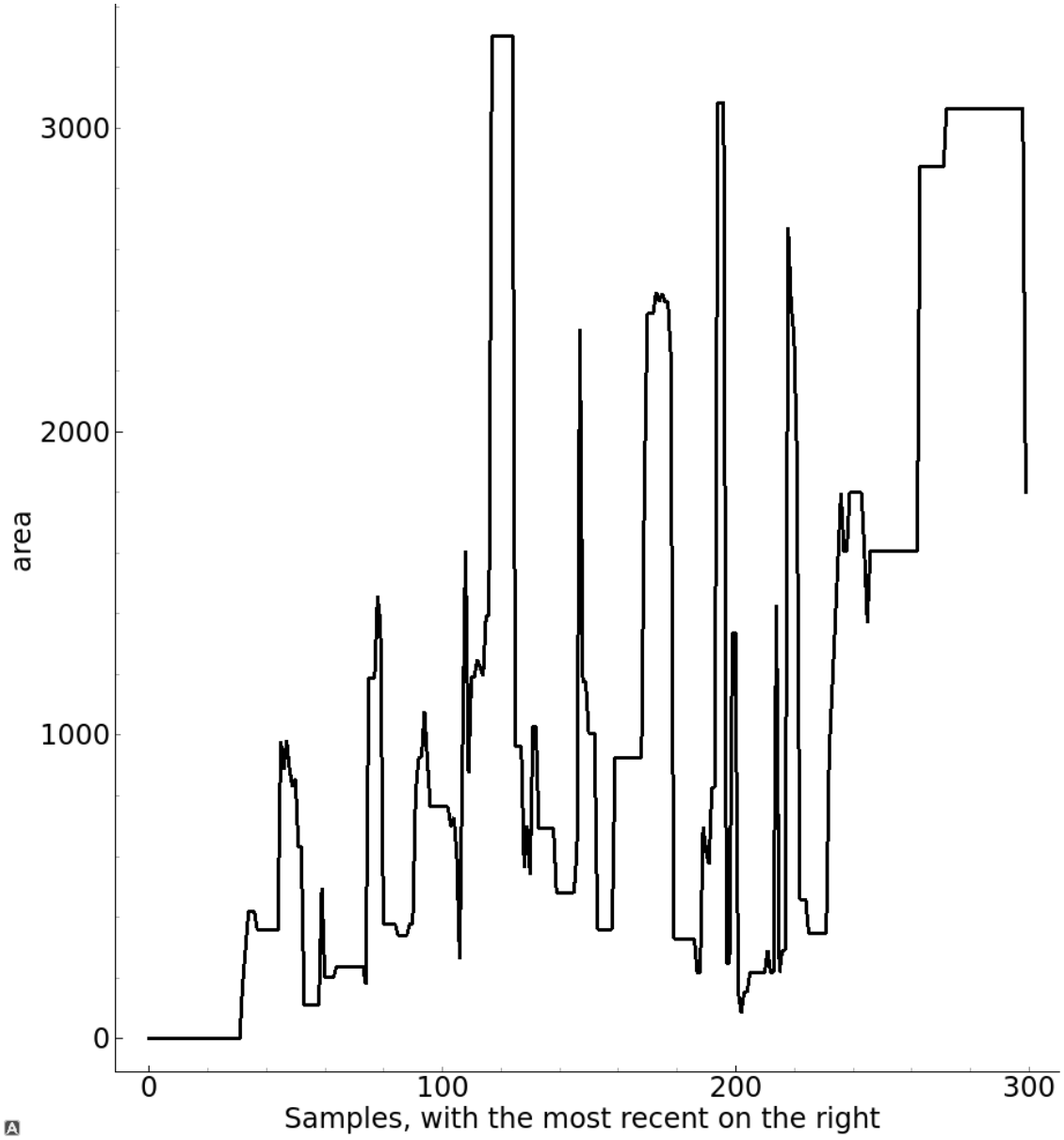


Figure A.2: Area of the largest group in squared centimetres over time. Data was captured during a person sitting down and standing up from a chair.

A.3. Area measurement of the largest group during a fall

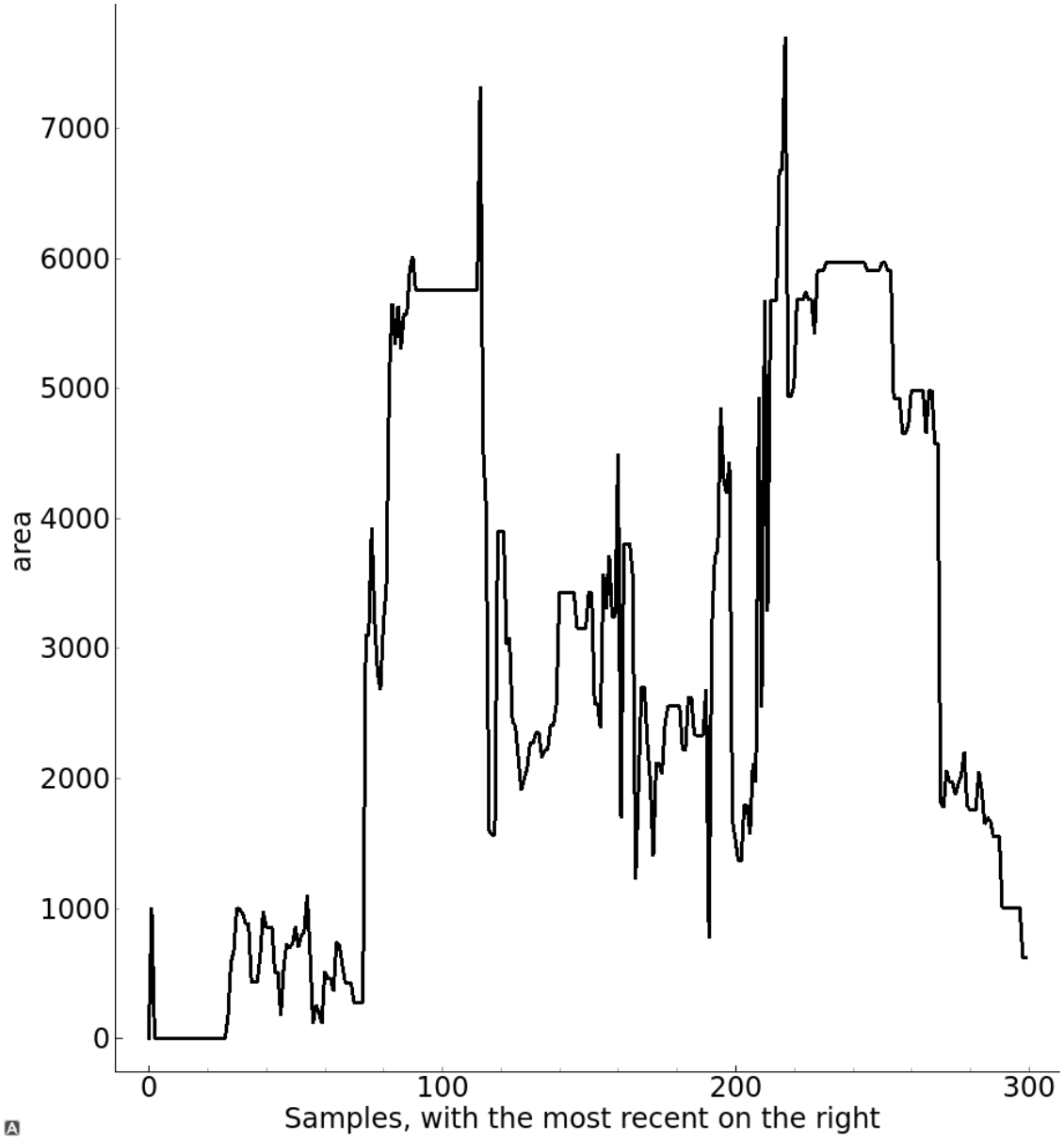


Figure A.3: Area of the largest group in squared centimetres over time. Data was captured during a person falling and taking various positions.

B
Length threshold measurements

B.1. Group length measurements

During the tests described in section 6.4, length in centimetres of the largest group in area was measured. The results can be seen here. The length was defined as the longest side of a bounding box around the group.

B.1.1. Length measurement of the largest group during walking

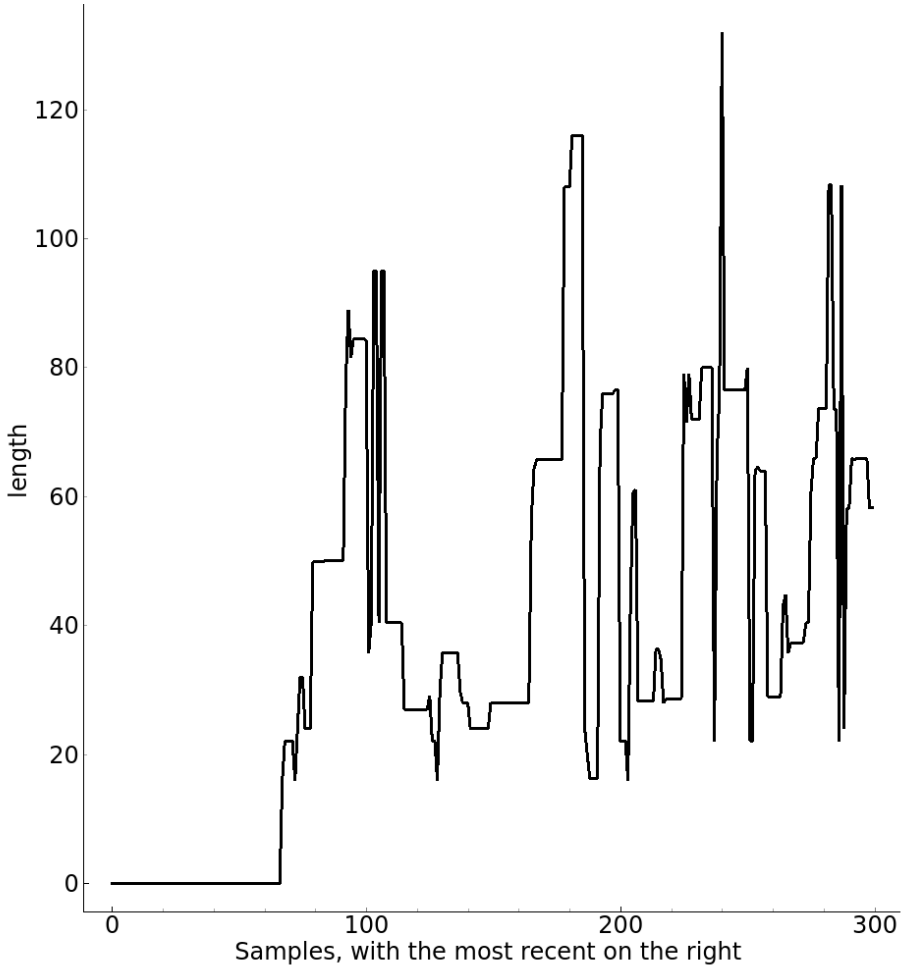


Figure B.1: Length of the largest group in centimetres over time. Data was captured during normal walking.

B.1.2. Length measurement of the largest group while sitting on a chair

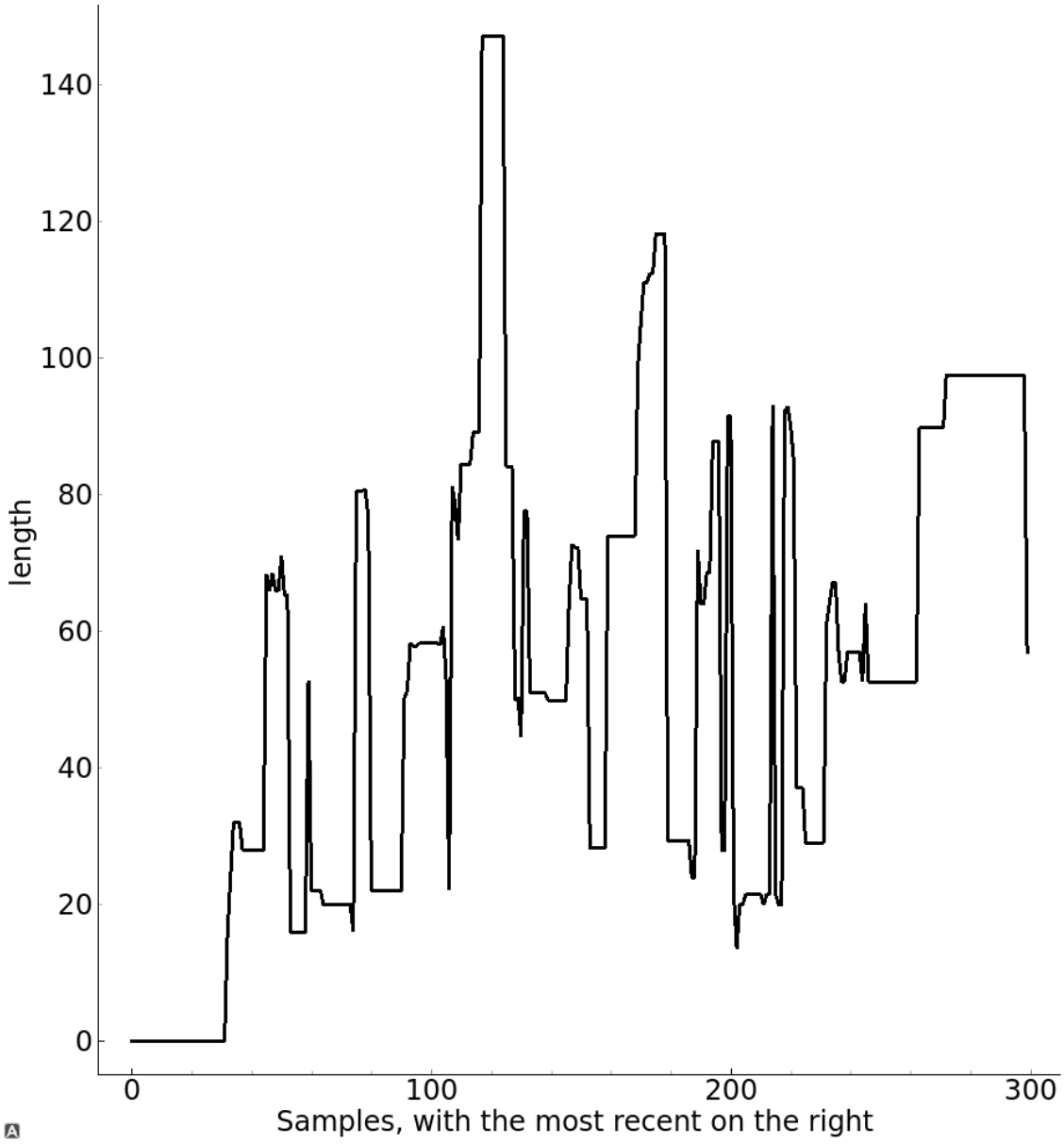


Figure B.2: Length of the largest group in centimetres over time. Data was captured during a person sitting down and standing up from a chair.

B.1.3. Length measurement of the largest group during a fall

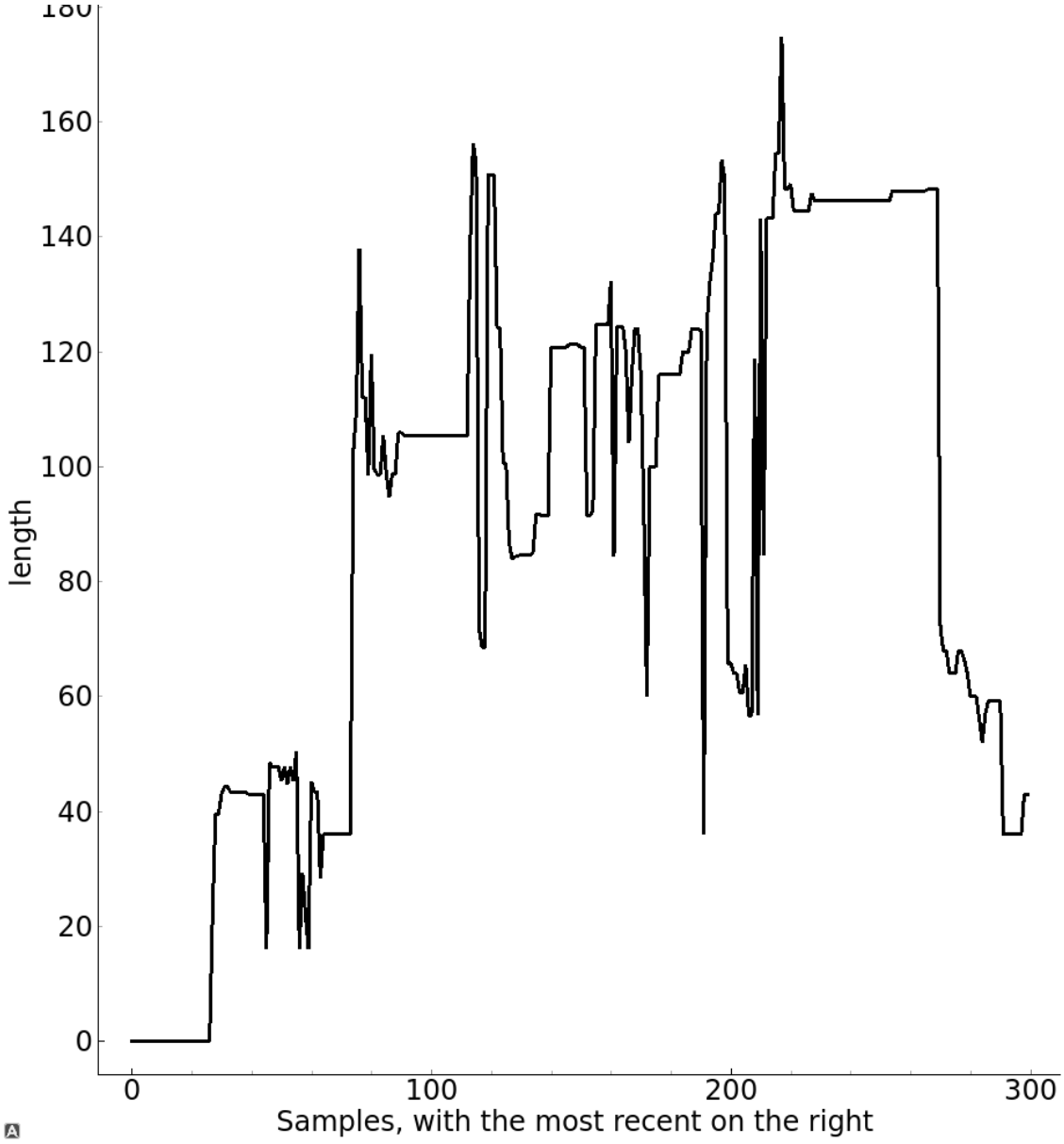


Figure B.3: Length of the largest group in centimetres over time. Data was captured during a person falling and taking various positions.

B.2. Contour length measurements

During the tests described in section 6.4, length in centimetres of the largest contour in area was measured. The results can be seen here. The length was defined as the longest side of a bounding box around the contours. Only the longest contour is shown at any moment.

B.2.1. Length measurement of the longest contour during walking

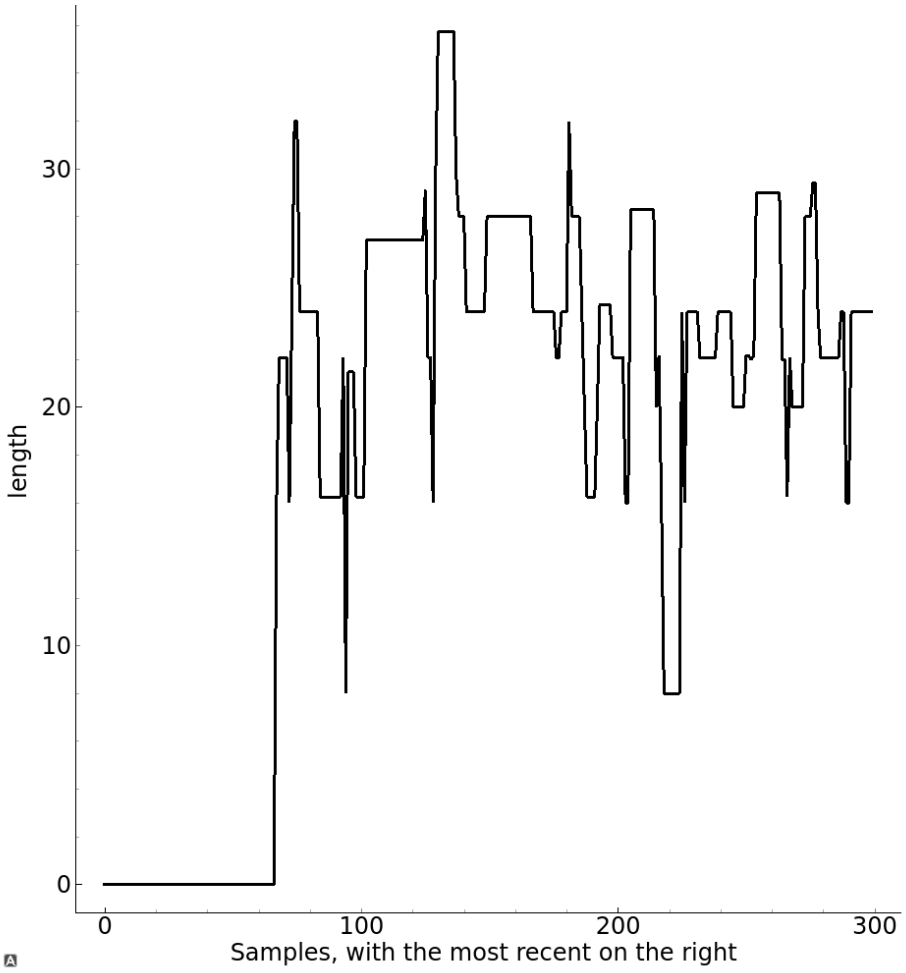


Figure B.4: Length of the longest contour in centimetres over time. Data was captured during normal walking.

B.2.2. Length measurement of the longest contour while sitting on a chair

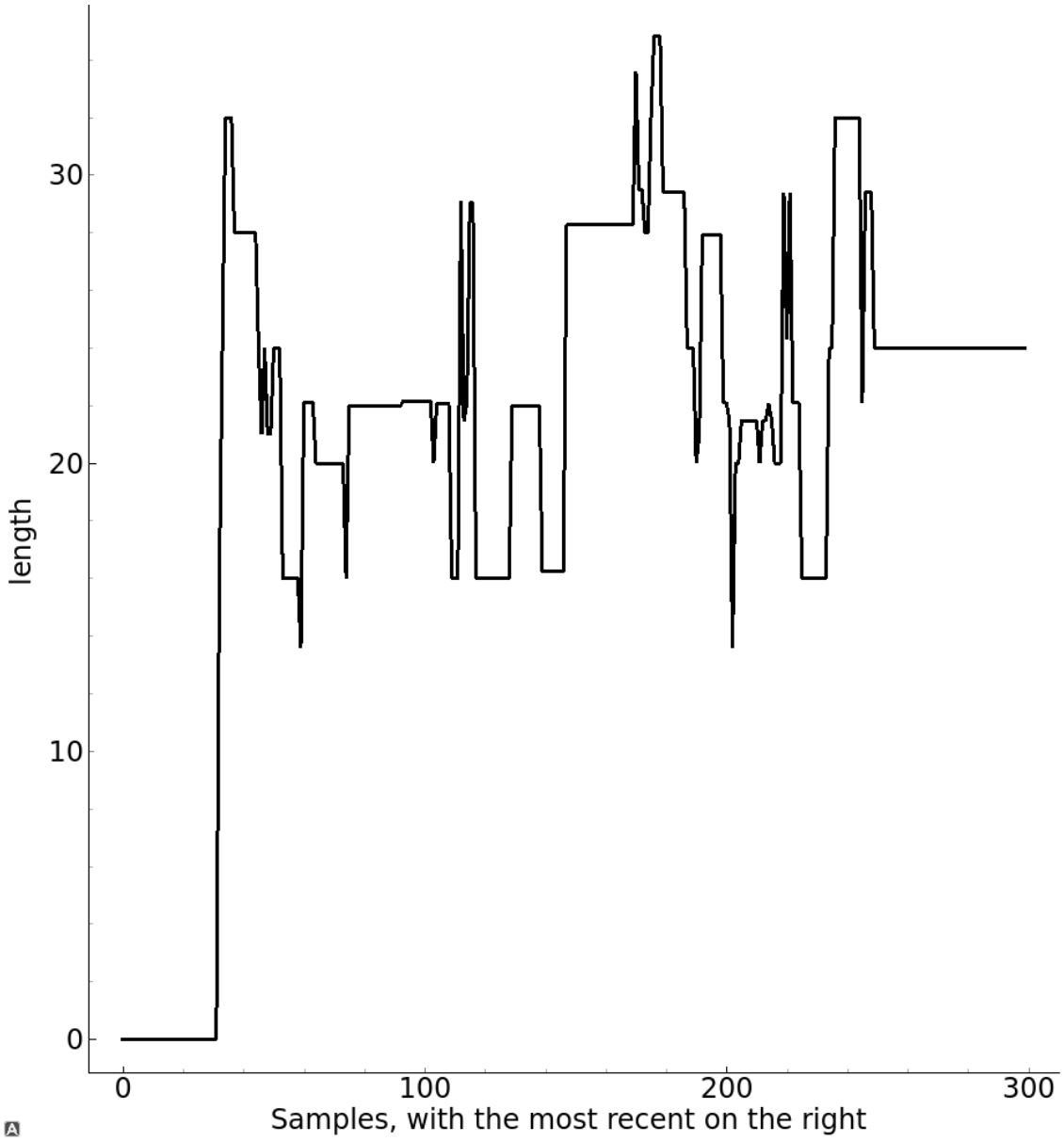


Figure B.5: Length of the longest contour in centimetres over time. Data was captured during a person sitting down and standing up from a chair.

B.2.3. Length measurement of the longest contour during a fall

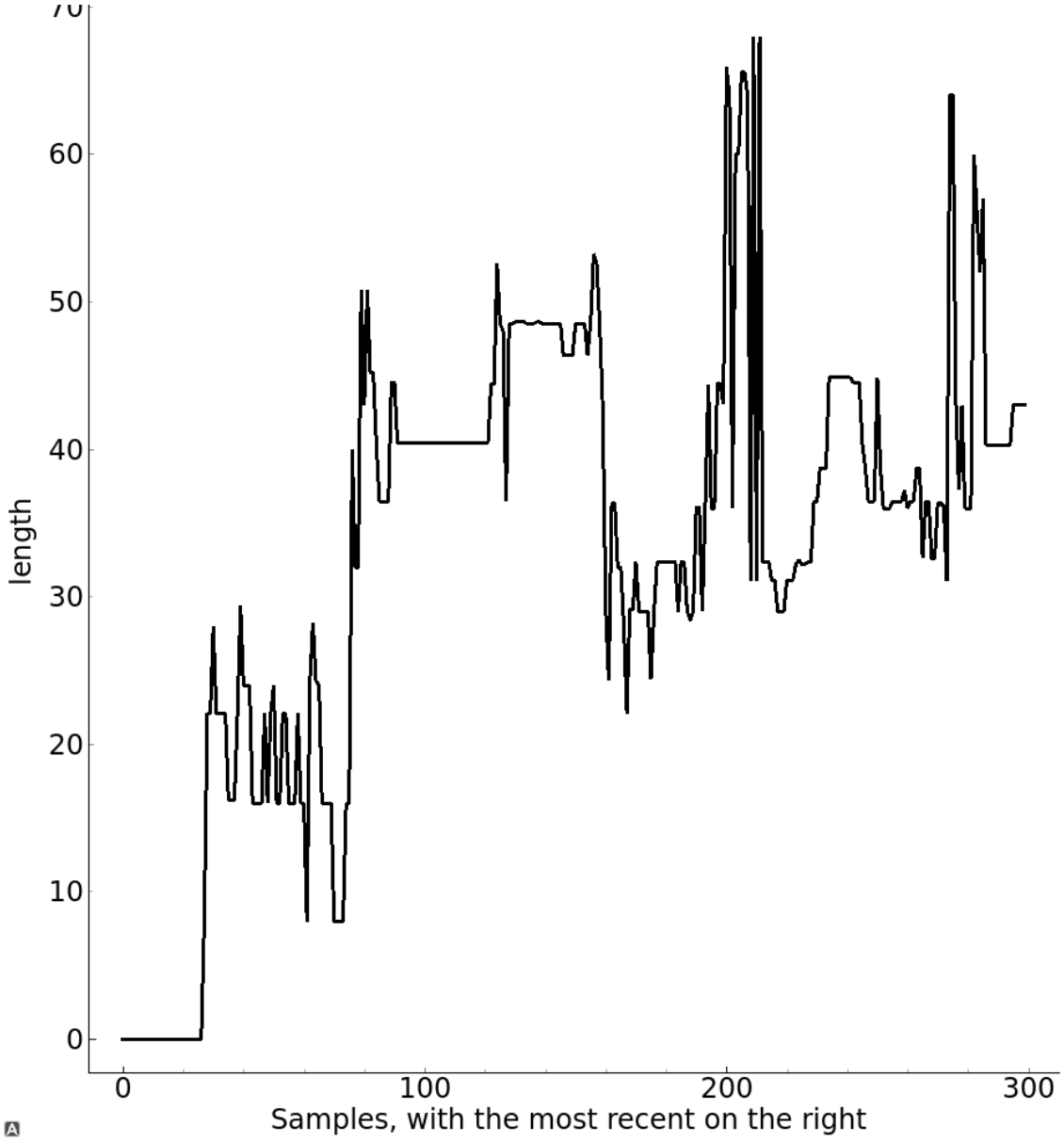


Figure B.6: Length of the longest contour in centimetres over time. Data was captured during a person falling and taking various positions.

C
Contour number measurements

During the tests described in section 6.4, the number of contours that make up the largest contour in terms of area was measured. The results can be seen here.

C.1. Contour count of the largest group during walking

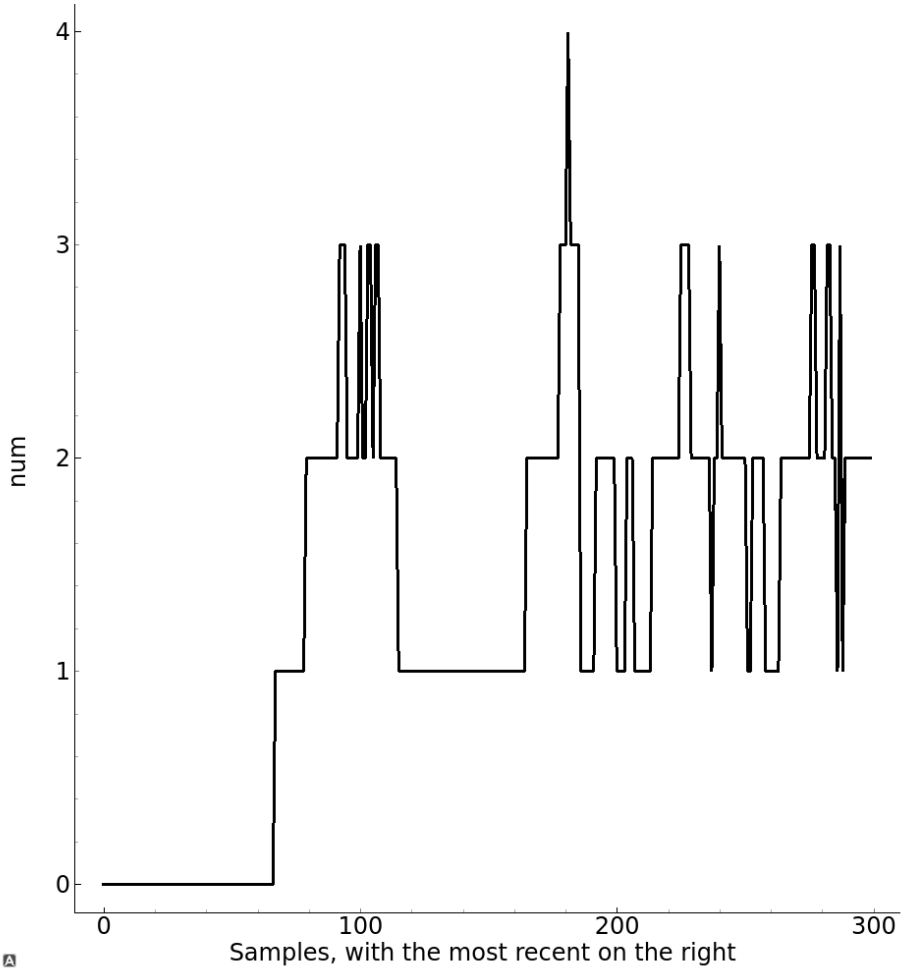


Figure C.1: The number of contours in the largest group over time. Data was captured during normal walking.

C.2. Contour count of the largest group while sitting on a chair

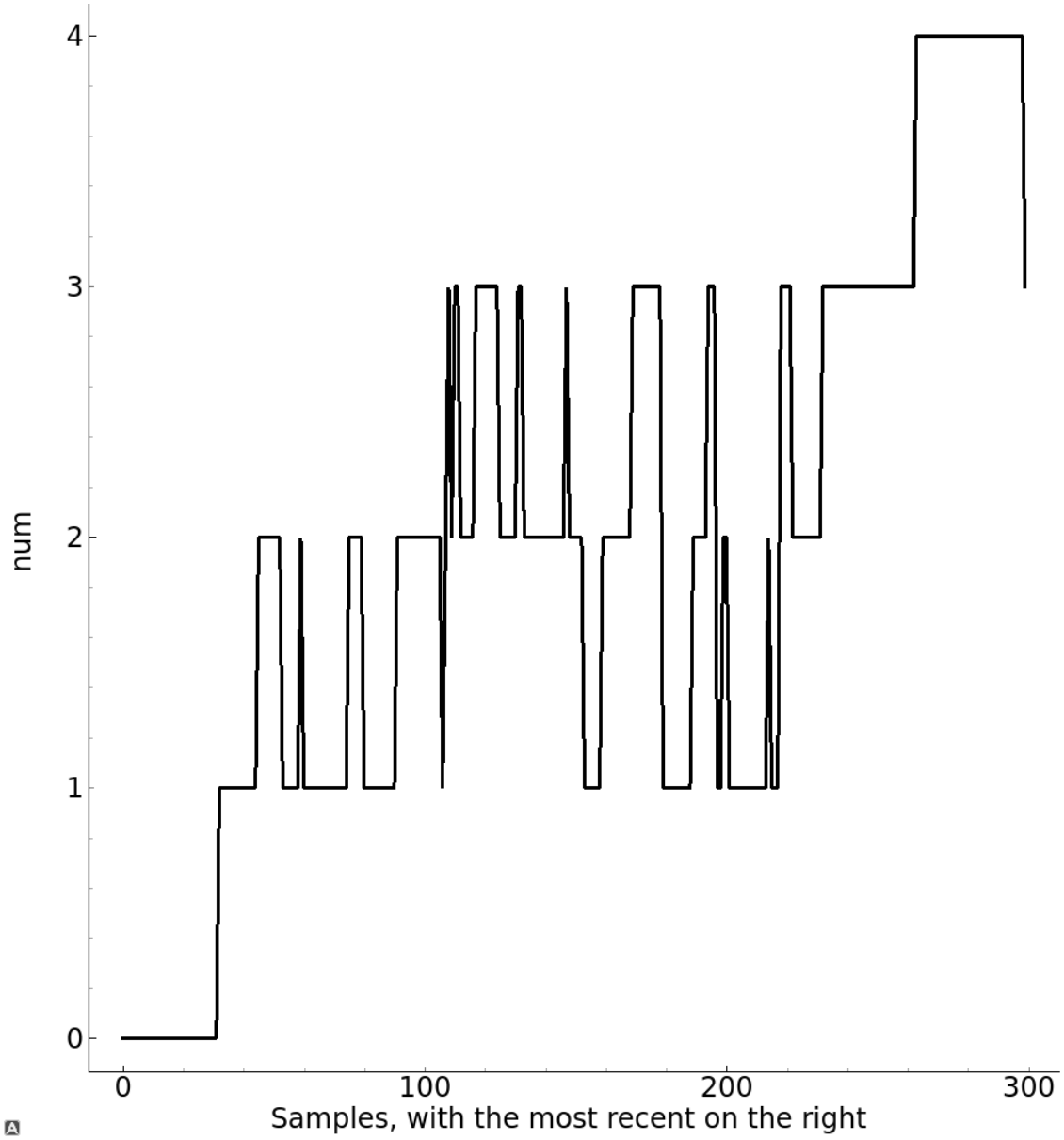


Figure C.2: The number of contours in the largest group over time. Data was captured during a person sitting down and standing up from a chair.

C.3. Contour count of largest group during a fall

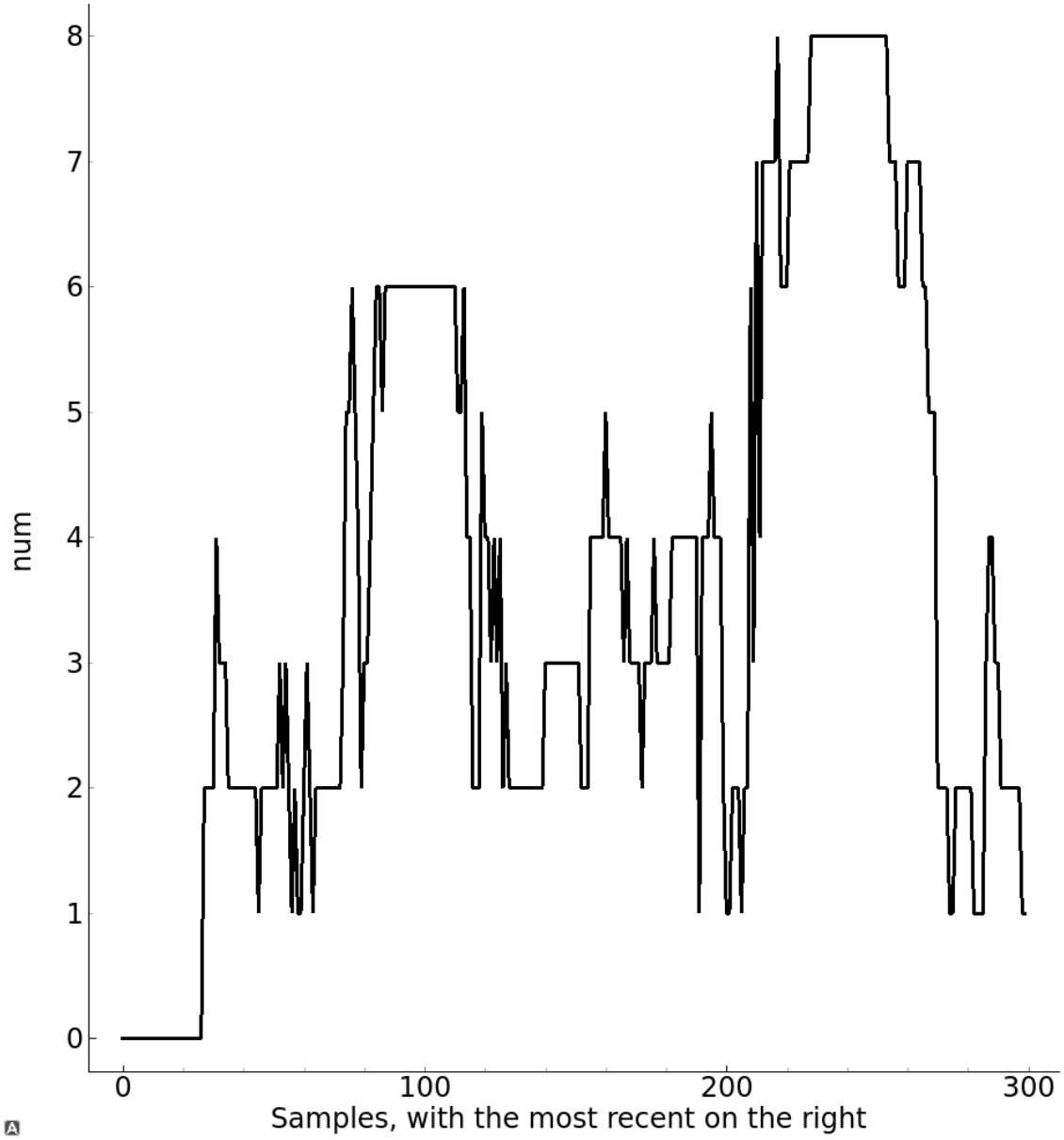


Figure C.3: The number of contours in the largest group over time. Data was captured during a person falling and taking various positions.

D

Aspect ratio measurements

During the tests described in section 6.4, aspect ratio of the largest contour in area was measured. The results can be seen here. Aspect ratio was defined as: shortest side divided by the longest side of a bounding box around the largest contour.

D.1. Aspect ratio measurement of the largest group during walking

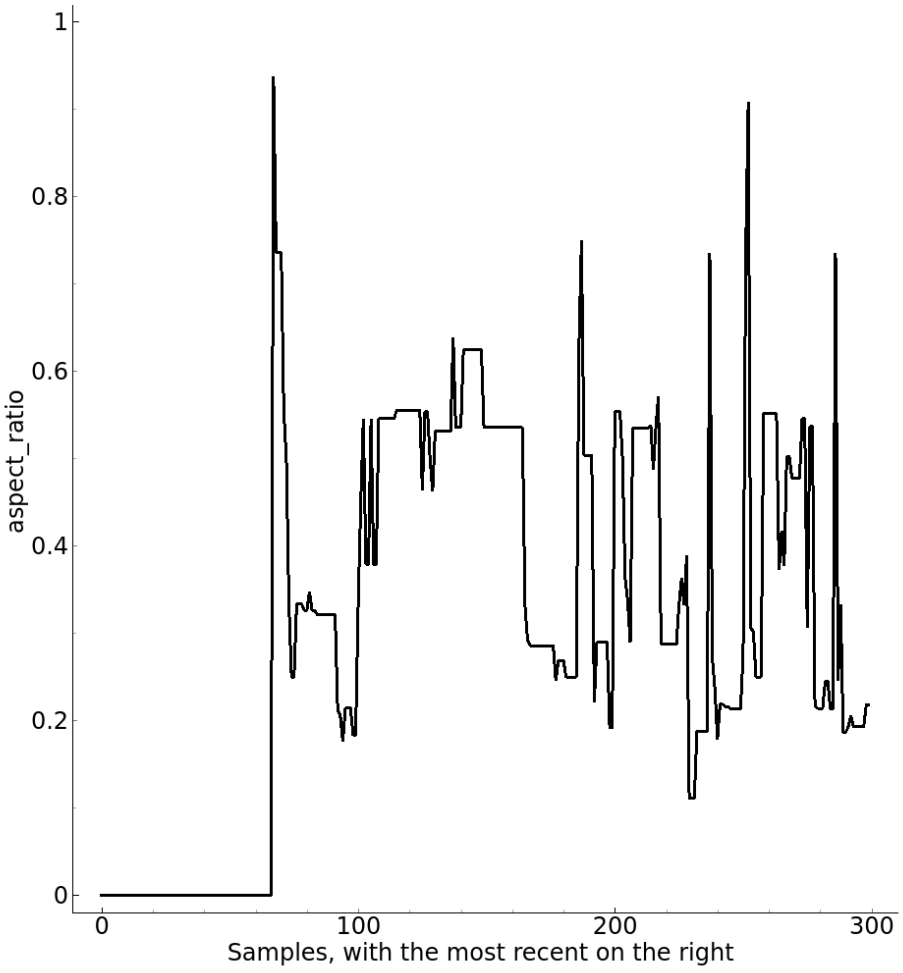


Figure D.1: Aspect ratio of the largest group over time. Data was captured during normal walking.

D.2. Aspect ratio measurement of the largest group while sitting on a chair

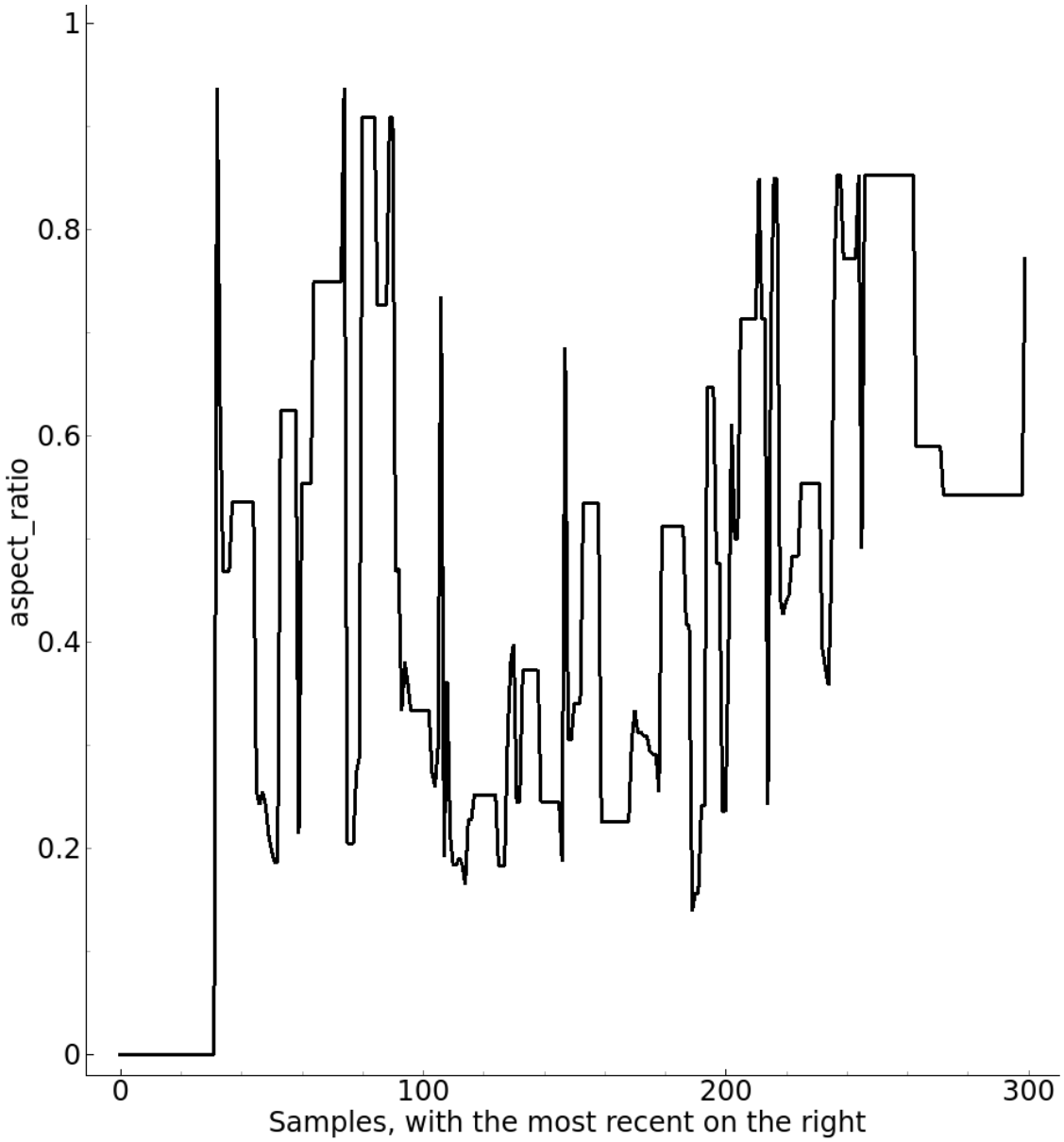


Figure D.2: Aspect ratio of the largest group over time. Data was captured during a person sitting down and standing up from a chair.

D.3. Aspect ratio measurement of the largest group during a fall

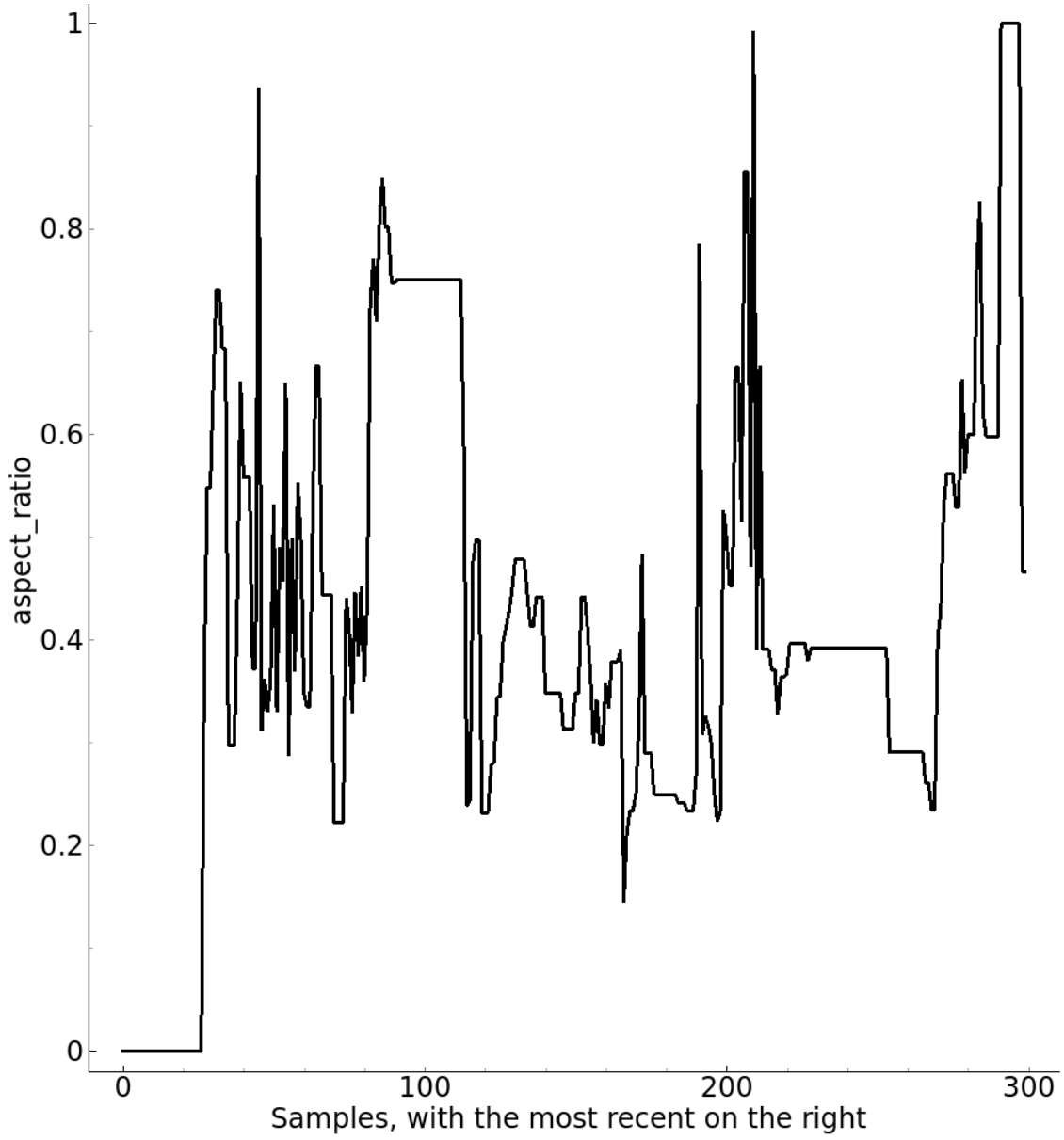


Figure D.3: Aspect ratio of the largest group over time. Data was captured during a person falling and taking various positions.

E

Weight measurements

During the tests described in section 6.4, weight of the largest contour in area was measured. The results can be seen here. Weight was defined as: sum over pixel values in the group scaled by a factor ($g \cdot 27/255$)

E.1. Weight measurement of the largest group during walking

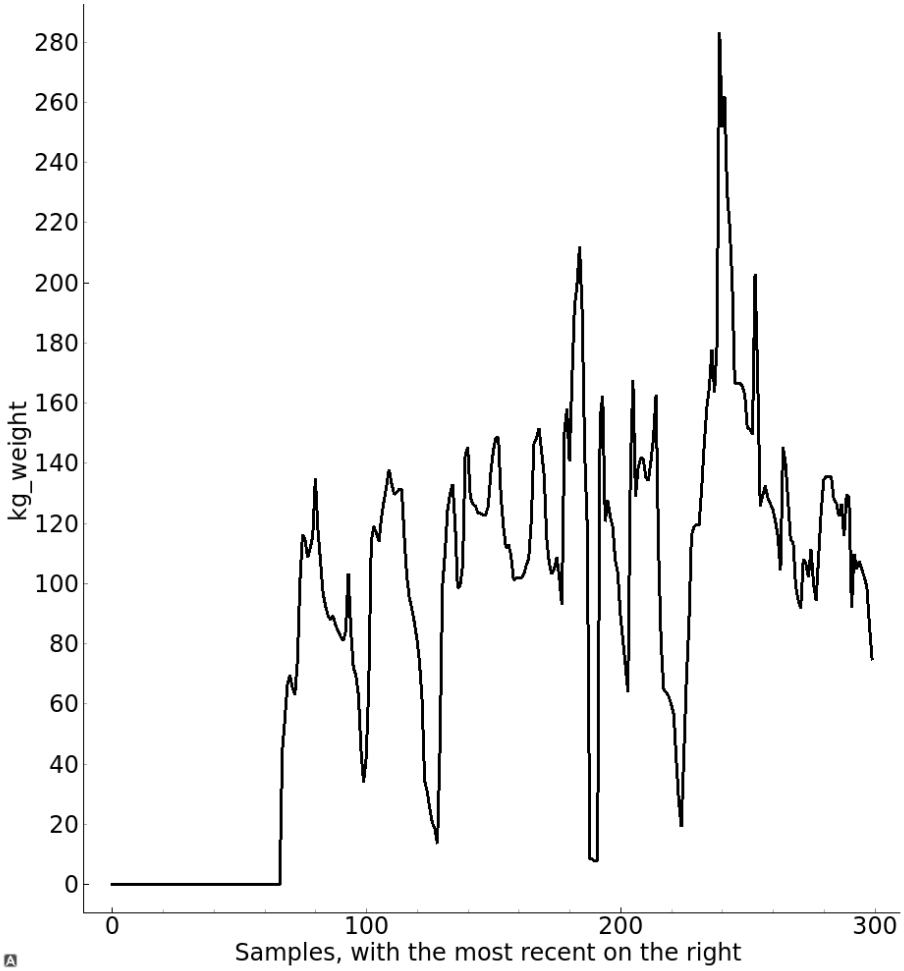


Figure E.1: Weight measurement of the largest group over time. Data was captured during normal walking.

E.2. Weight measurement of the largest group while sitting on a chair

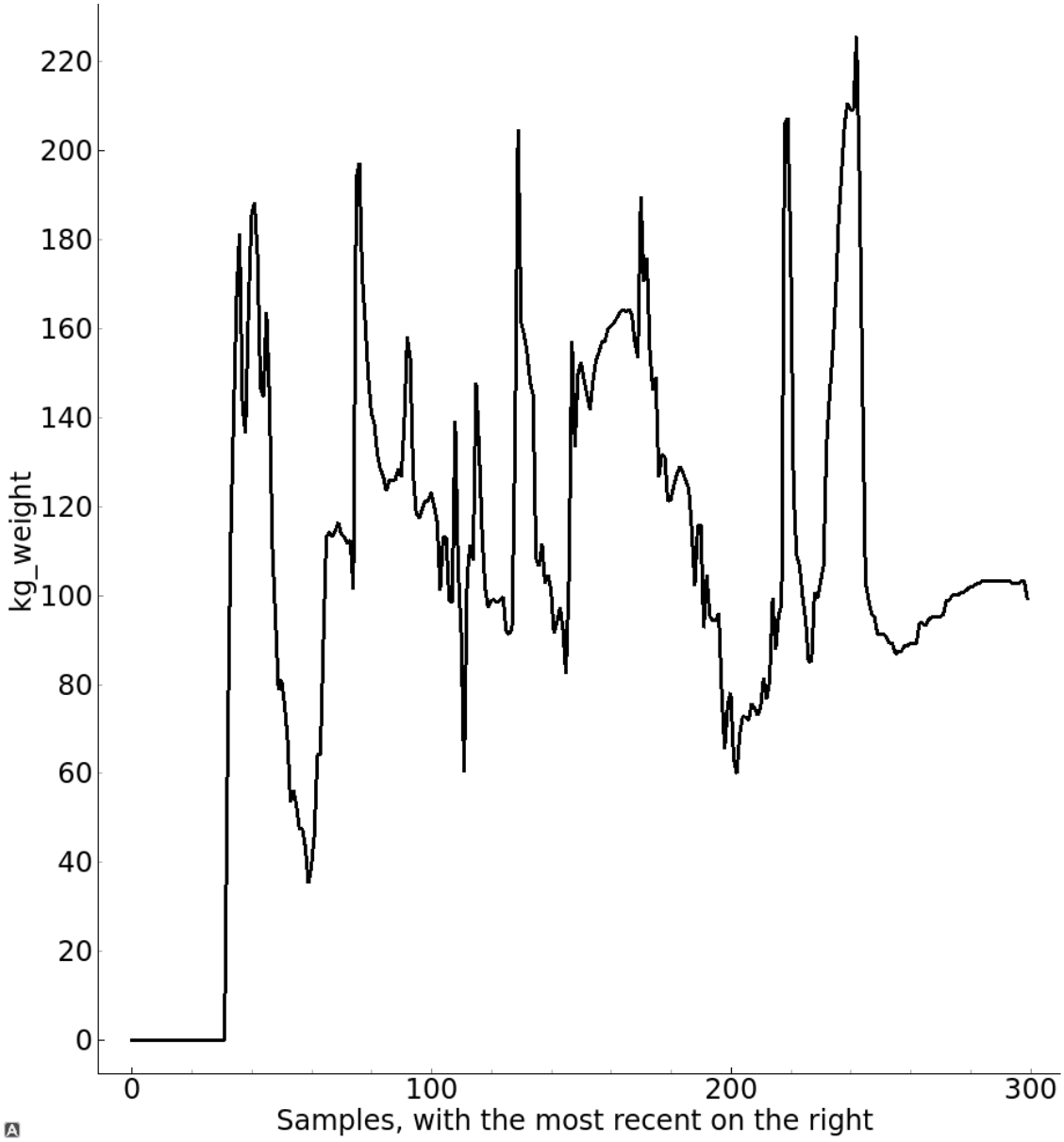


Figure E.2: Sum of intensity of the largest group over time. Data was captured during a person sitting down and standing up from a chair.

E.3. Weight measurement of the largest group during a fall

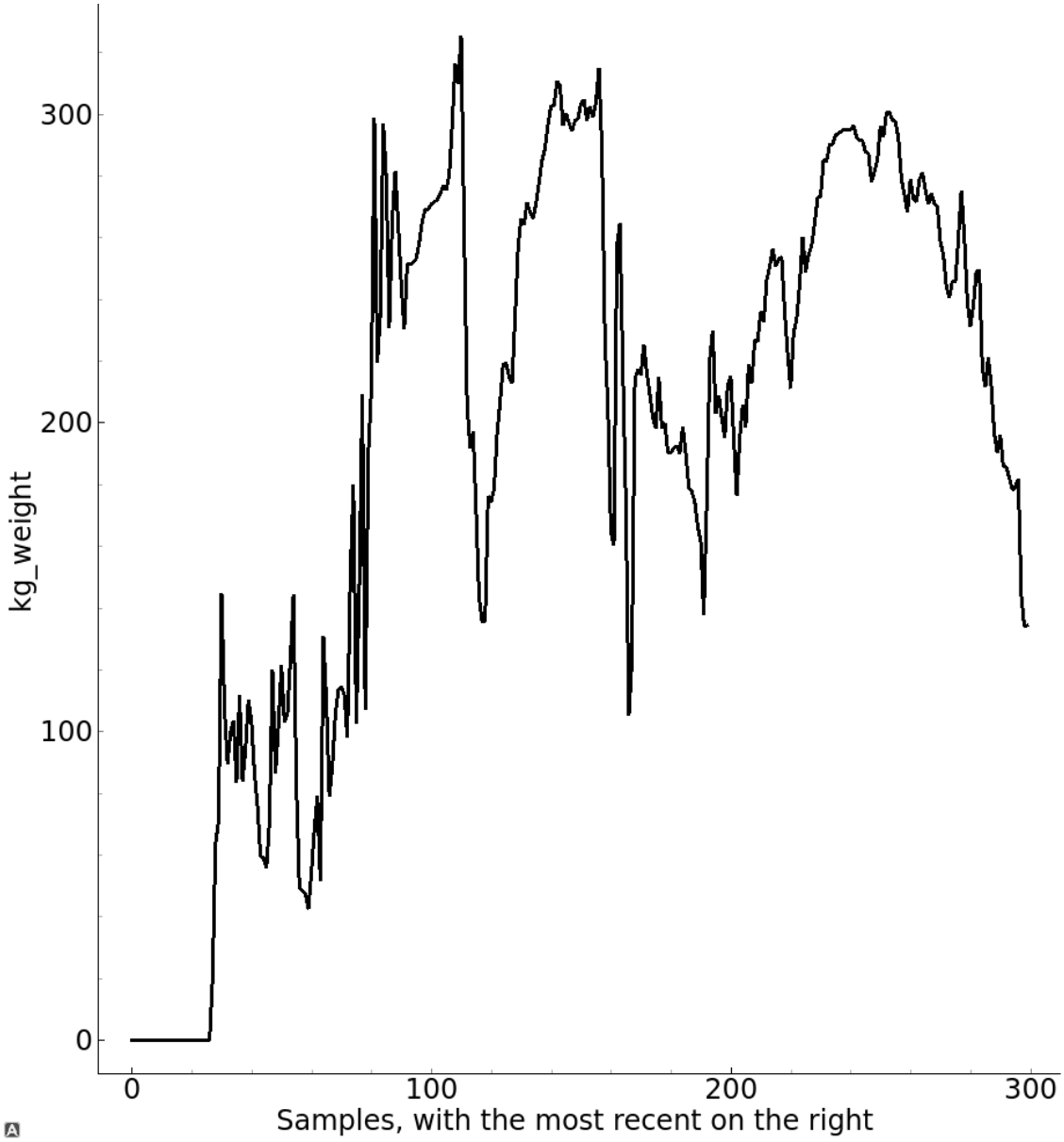


Figure E.3: Weight measurement of the largest group over time. Data was captured during a person falling and taking various positions.

F

Average pressure measurements

During the tests described in section 6.4, average pressure of the largest contour in terms of area was measured. The results can be seen here. Average pressure was defined as the sum over pixel values in the group scaled by a factor $(27/(255 \cdot g))$ to convert to kg. This value was then divided by the area of the group.

F.1. Average pressure measurement of the largest group during walking

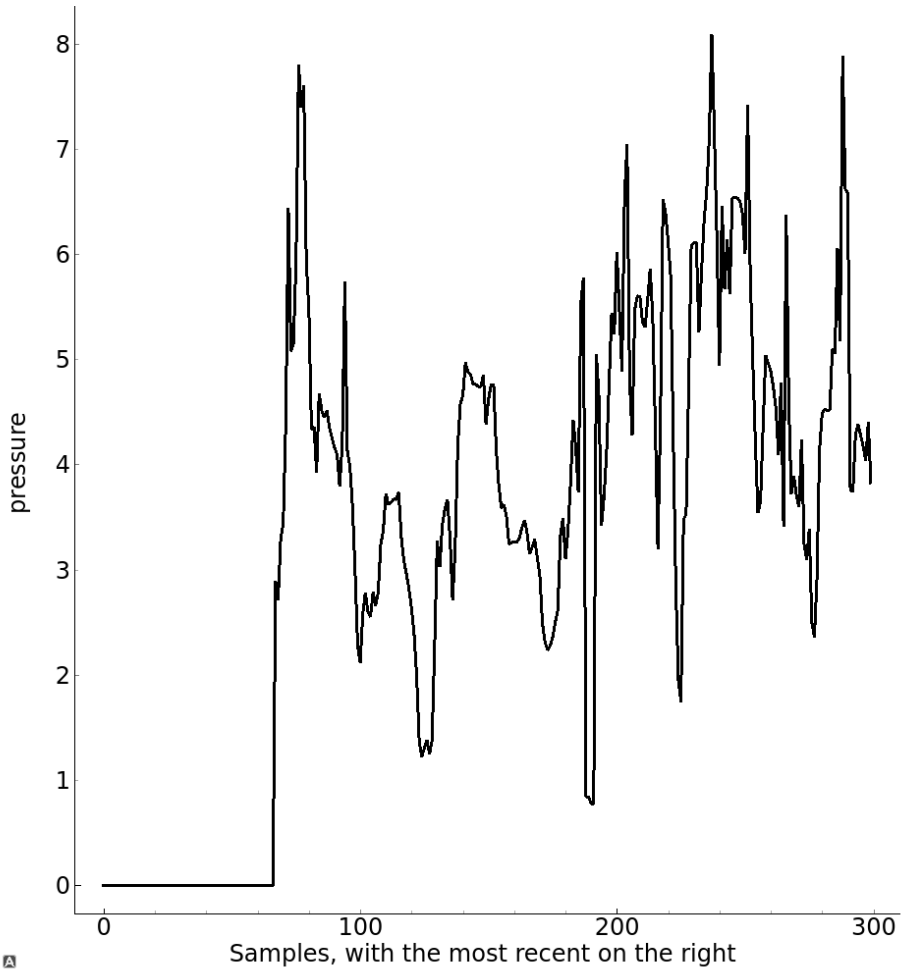


Figure F.1: Pressure of the largest group over time. Data was captured during normal walking. Data is converted to units of N/cm^2

F.2. Average pressure measurement of the largest group while sitting on a chair

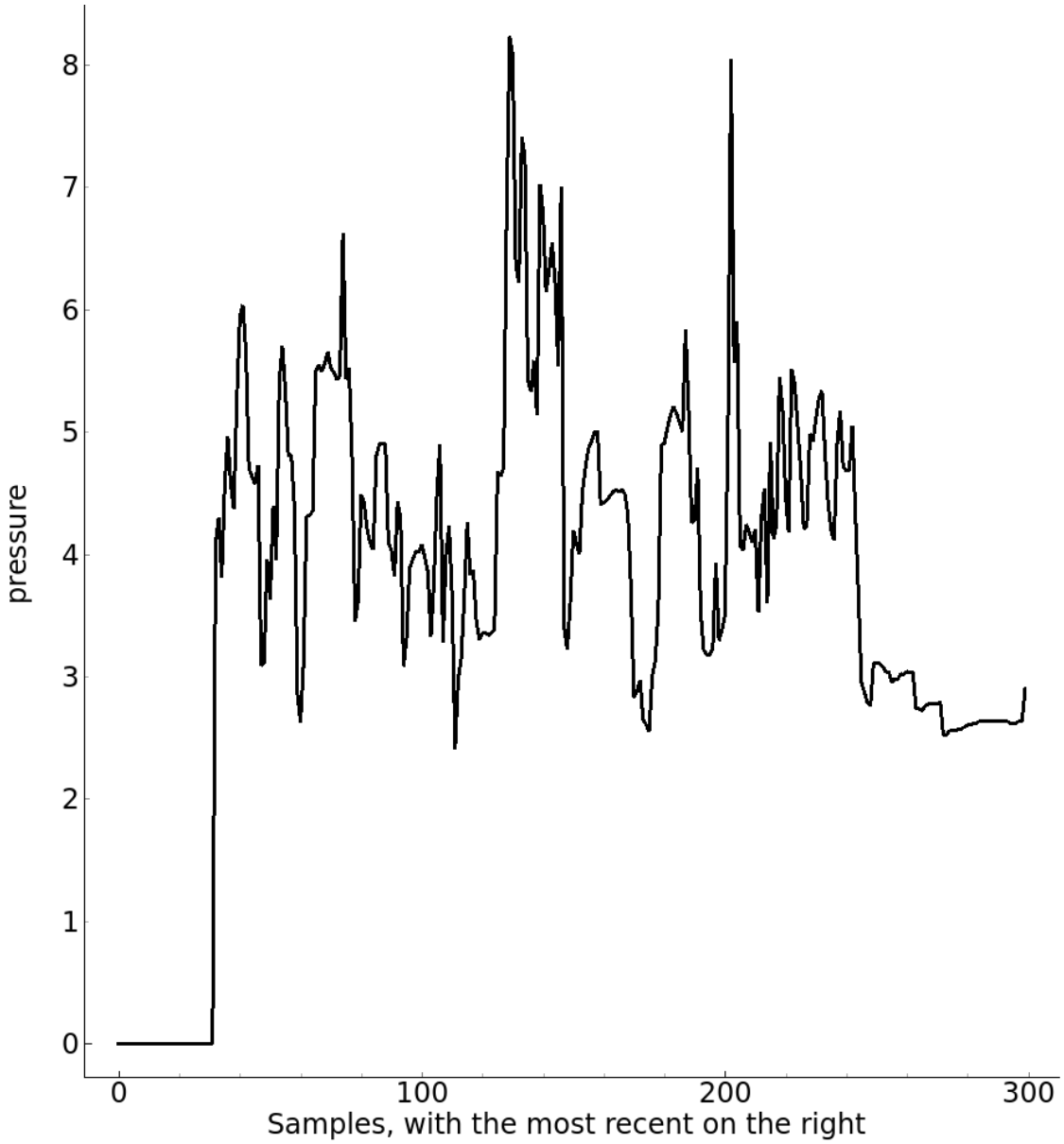


Figure F.2: Pressure of the largest group over time. The data represents a person sitting down and standing up from a chair. Data is converted to units of N/cm^2

F.3. Average pressure measurement of the largest group during a fall

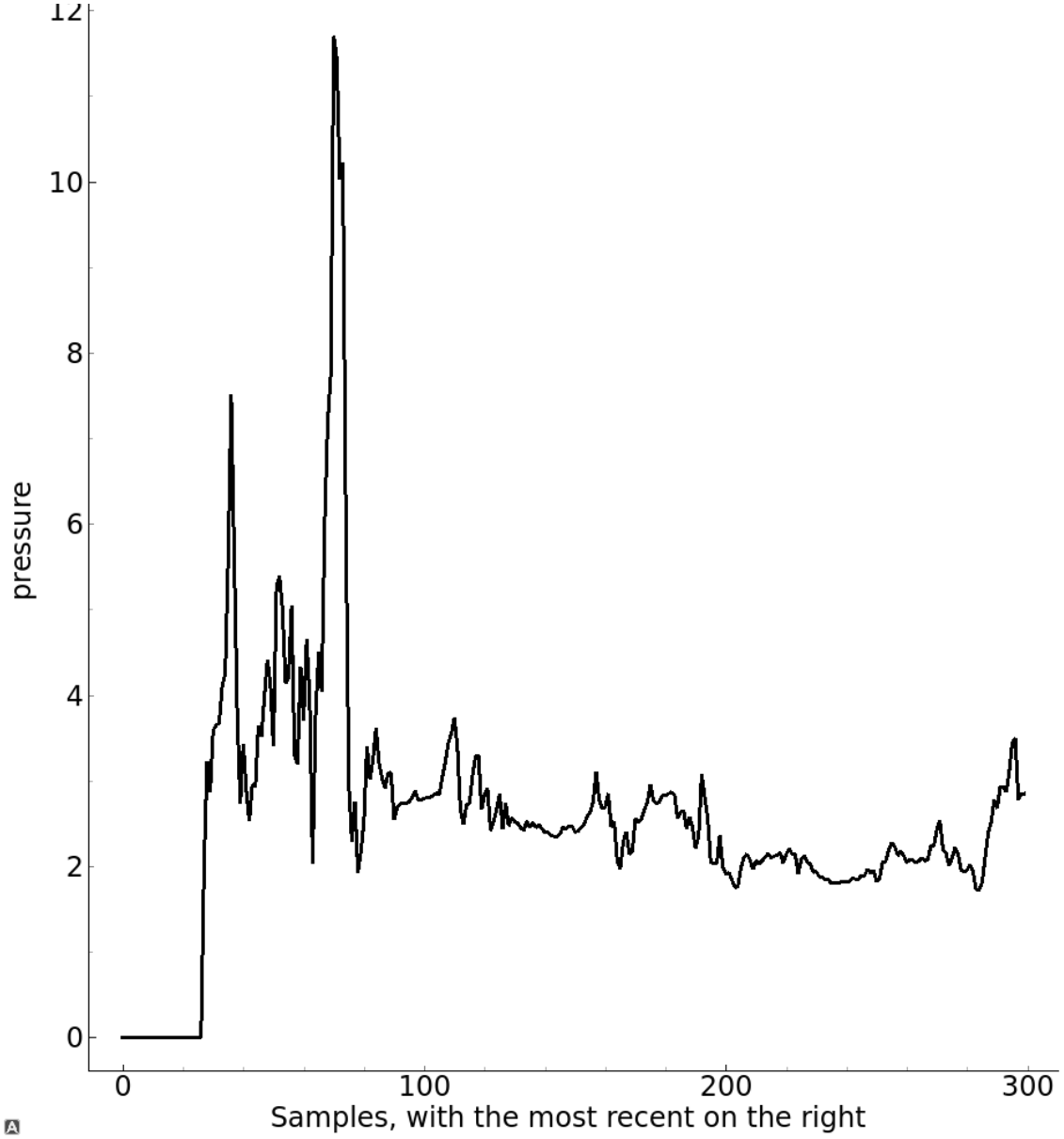


Figure F.3: Pressure of the largest group over time. Data was captured during a person falling and taking various positions.

G
Maximum pressure measurements

During the tests described in section 6.4, maximum pressure of the largest contour in area was measured. The results can be seen here. Maximum pressure was defined as: highest pixel value in the group scaled by a factor ($g \cdot 27/255$).

G.1. Maximum pressure measurement of the largest group during walking

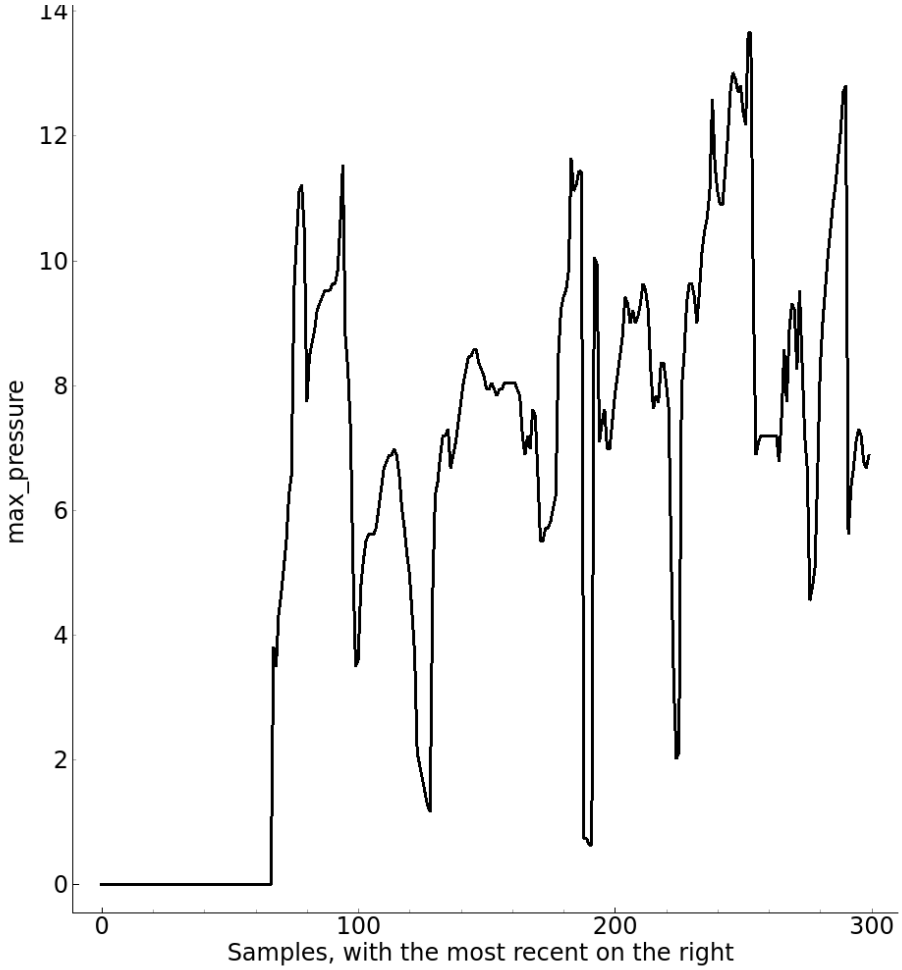


Figure G.1: Pressure of the largest group over time. Data was captured during normal walking. Data is converted to units of N/cm^2

G.2. Maximum pressure measurement of the largest group while sitting on a chair

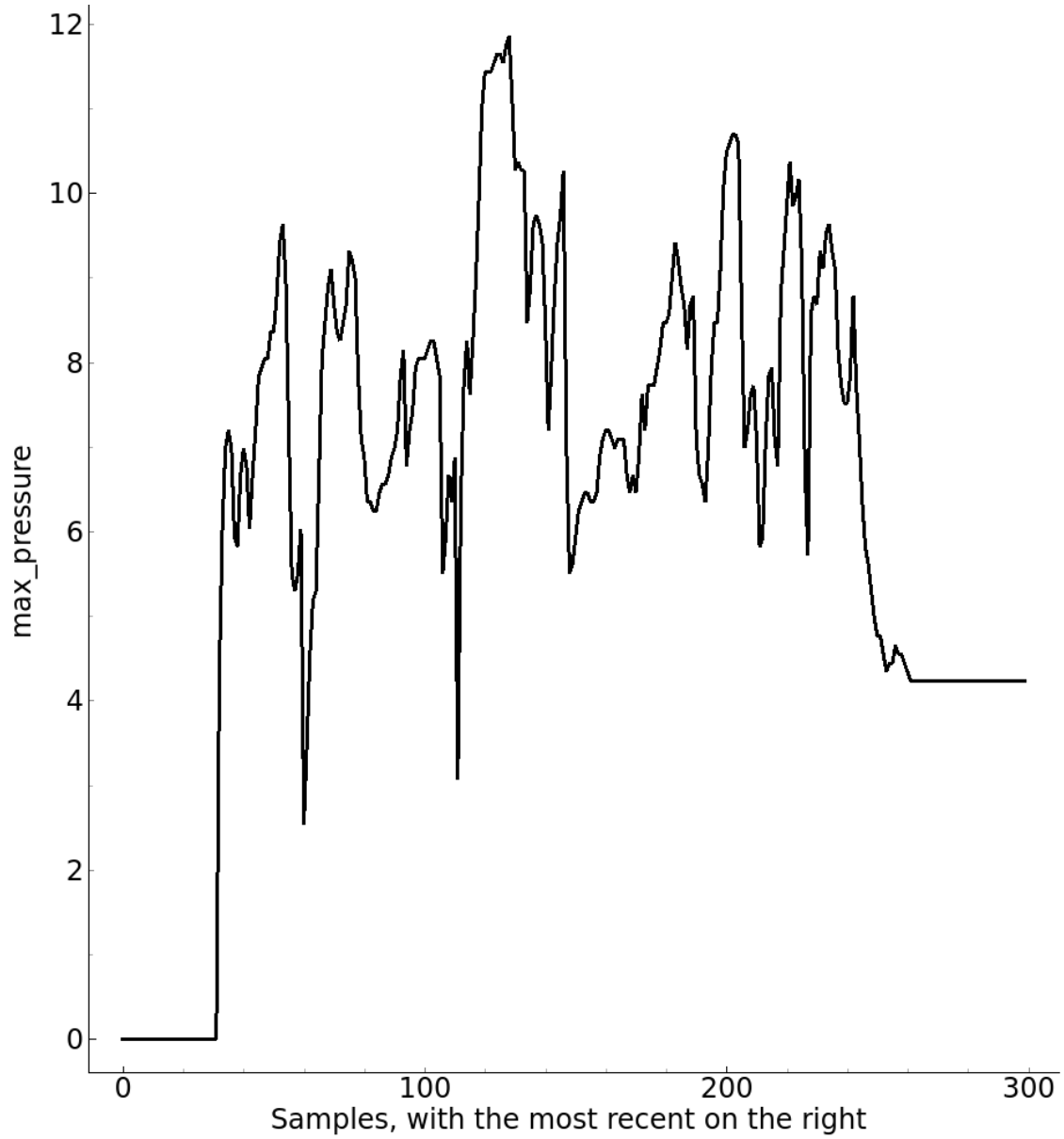


Figure G.2: Pressure of the largest group over time. The data represents a person sitting down and standing up from a chair. Data is converted to units of N/cm^2

G.3. Maximum pressure measurement of the largest group during a fall

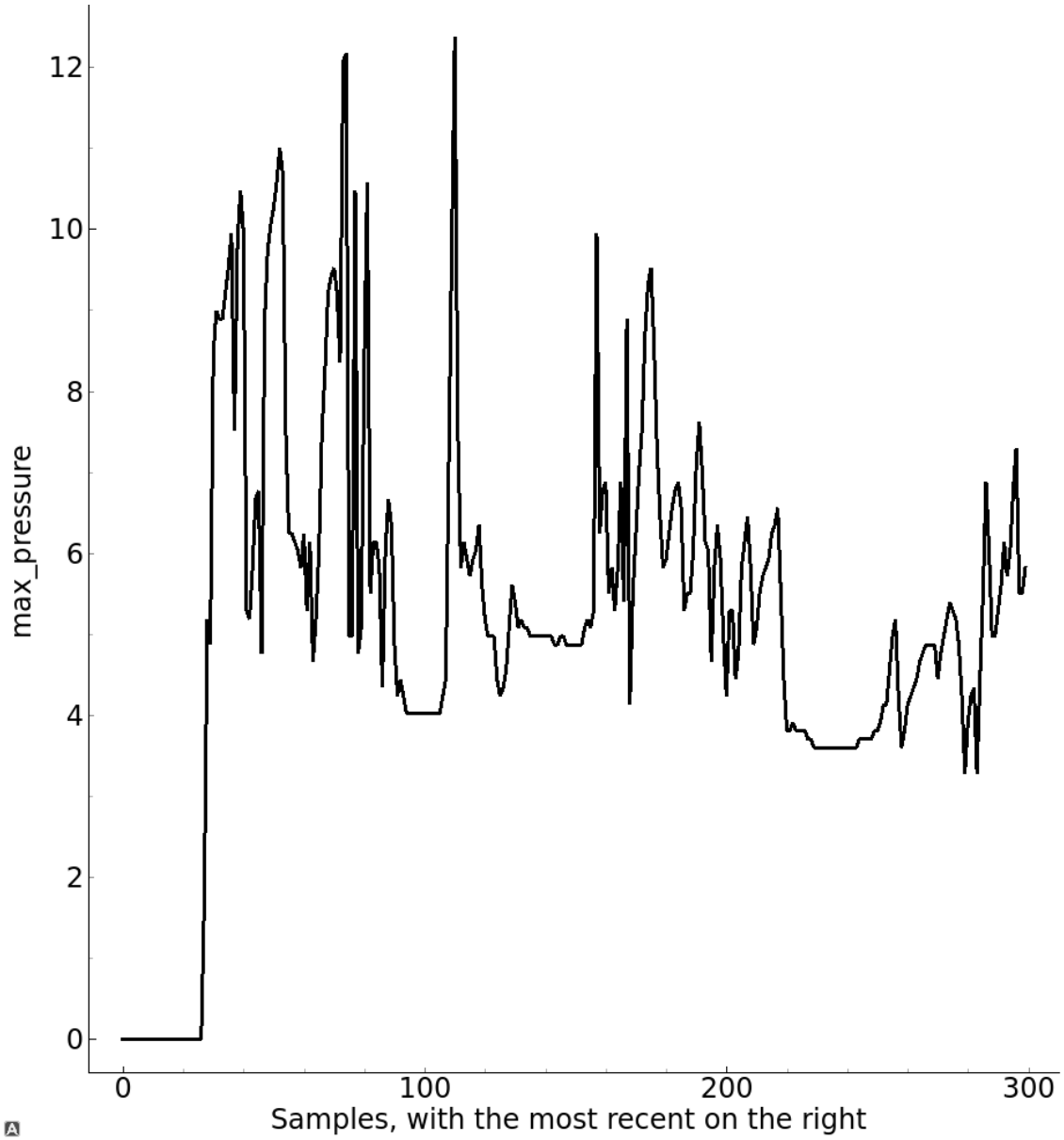


Figure G.3: Pressure of the largest group over time. Data was captured during a person falling and taking various positions.

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] R. Johansson, *Numerical Python*. Apress, 01 2019.

[32] A. R. K. Alexander Mordvintsev, "Introduction to opencv-python tutorials," 2019, accessed: 11-6-2019. [Online]. Available: www.docs.opencv.org/3.0-beta/doc/py_tutorials/py_setup/py_intro/py_intro.html#intro

[33] M. Birtane and H. Tuna, "The evaluation of plantar pressure distribution in obese and non-obese adults," *Clinical Biomechanics*, vol. 19, no. 10, pp. 1055 – 1059, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0268003304001627

[34] F. Pu, Y. Yang, X. Fan, S. Li, Y. Li, D. Li, and Y. Fan, "Optimal estimation of total plantar force for monitoring gait in daily life activities with low-price insole system," *Journal of Mechanics in Medicine and Biology*, vol. 14, 03 2014.

[35] G. Bradski and A. Kaehler, *Learning OpenCV 3*. O'Reilly Media, Inc., 2016.

[36] L. H. L. Cihalova, "Impact injury prediction by fe human body model," *Applied and Computational Mechanics*, 2008.