Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Institute of Neuroinformatics

Footstep Classification

René Beutler
Semester Report

February 2001

Professor: Prof. R. Douglas
Mentor: Tobi Delbruck

Abstract

Ada is a project designated to build an intelligent space. Intelligent means that the space is able to sense its visitors by visual, auditory and force information, has a personality, expresses its different emotional states with light and sounds, and actively interacts with individuals and groups.

One of Ada's senses is supplied by an active sensory floor. The floor has a regular structure based on hexagonal floor tiles. Every floor tile is equiped with load sensors which not only allow to locate people on the floor but also provide a clue about the walking style or the weight of a person possibly standing on the tile.

The variation of the output among different floor tiles is so considerable that without any further preprocessing the only statement that can be made is whether the tile is occupied by a person or not.

The goal of this work is to present an efficient mechanism which yields more meaningful data as a basis for further, more high-level processing. The task is to perform an online calibration of the floor sensors. It will be shown that the problem can be solved by using freely walking people as a reference; no testweights have to be placed on the floor.

The fundamental idea is to compare the outputs of tiles that are related to a common person, respectively a common force. Therefore a lot of effort will be put into the recognition of traces of a walking person. From a model of the sensor variability a method to calibrate the floor tile output will be derived.

The presented algorithms are implemented in form of a real time standalone program that was tested on a small prototype floor. Only a few steps on each tile are sufficient to obtain good calibration results which allows easy segmentation of children from adults.

Contents

1 Introduction

1.1 Ada - The context

At the Neuchâtel Arteplage of the swiss national exhibition Expo.02[1] a team of scientists from The Institute of Neuroinformatics[2], together with their collaborators, will present the futuristic project "ADA: The Intelligent Space"[3].

Ada is a room, named after Ada Byron King (one of the 19th century pioneers of computer science), that tries to interact with its visitors, it is curious and wants to play with them. In order to do that she has to be able to perceive and to feel the people inside. The room is therefore equipped with sensor devices to collect not only visual and auditory information, but also to have a sense of touch.

1.2 The active sensory floor

There are numerous visitors in the space at the same time. All visitors are different and Ada tries to distuingish them. She might show how she distinguishes someone by surrounding him with an own personal rippling pattern of colors emitted by the floor tiles. For example, Ada could adapt the way she interacts with someone to the age of the person.

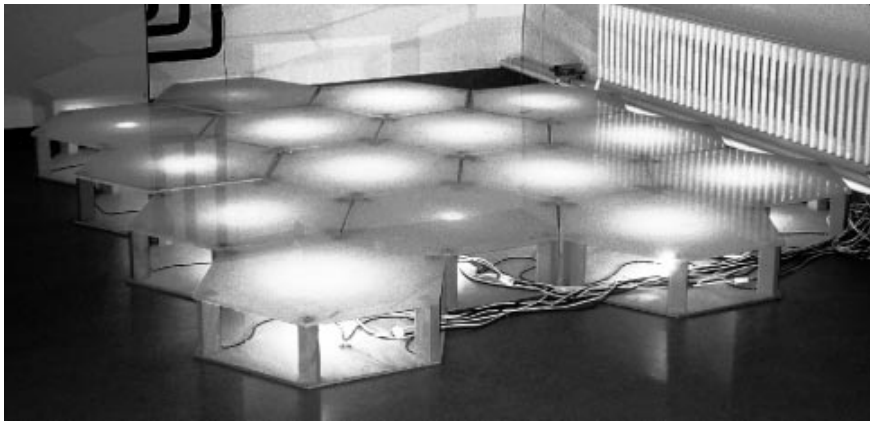


Figure 1: The Prototype active sensory floor.

The active floor is equipped with load sensors. This allows not only knowing the position where a person is standing, but also getting an idea of the persons weight, which helps with identifying or tracking people.

1.3 The sensor data and its problems

Figure 2 illustrates what the raw data coming from the floor looks like. One might guess that these are steps from two or three persons, but they all belong to the same person walking on three tiles.

The most important reason for this phenomenon is the variability of the force sensitive sensors. The conductance of two sensors can differ remarkably; additionally they seem to change their output over time.

This makes it impossible to use the data directly except from detection of tile occupation. Some calibration has to take place first.

[1] `http://www.expo-01.ch/`
[2] `http://www.ini.unizh.ch/`
[3] `http://zig.ini.unizh.ch/~expo/`

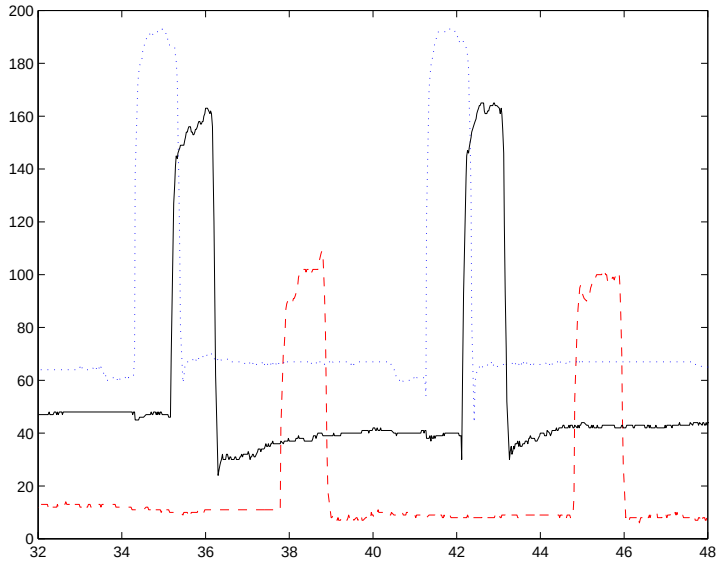


Figure 2: Output from three tiles, the same person stepping on each twice. Note the big variance.

1.4 The challenge

The goal of this work is to find a mechanism which only needs to be fed with raw data and yields more meaningful data as a basis for further, more high-level processing, like IQR421[4].

The task therefore is to perform an online calibration of the floor sensors, allowing a stable weight estimation of people. No known testweight must be used since it is not feasible to manually calibrate hundreds of tiles. Instead, freely walking people shall be used as reference.

1.5 Approach - An outline

The basic idea is to use visitors as live test weights that will successively cover the floor with reference values that can be compared amongst the floor tiles. The visitors weight is assumed not to change while walking on the floor. Therefore it is possible to directly compare the output of floor tiles that belong to the same person.

First, a model for the floor tile and its variability will be given. As it will be shown, one parameter per floor tile will suffice to describe the variability and to calibrate the whole floor. The goal is to find as many steps as possible that belong to the same person. On one hand, this allows the comparision of floor sensors, on the other hand the real weight of the person can be estimated more precisely the more data about it can be collected.

In section 2 a model for the force sensitive resistor (FSR) and the floor tile output will be given. These models are the basis of the calibration, as described in section 3. Section 4 describes how footsteps and moves are extracted from floor data. All the steps from the same person are collected in traces (section 5).

[4]IQR421 is a tool for the development and control of realistic large-scale real-time neuronal systems, which has been developed at the Institute of Neuroinformatics.

2 Models

2.1 Force Sensitive Resistor Model

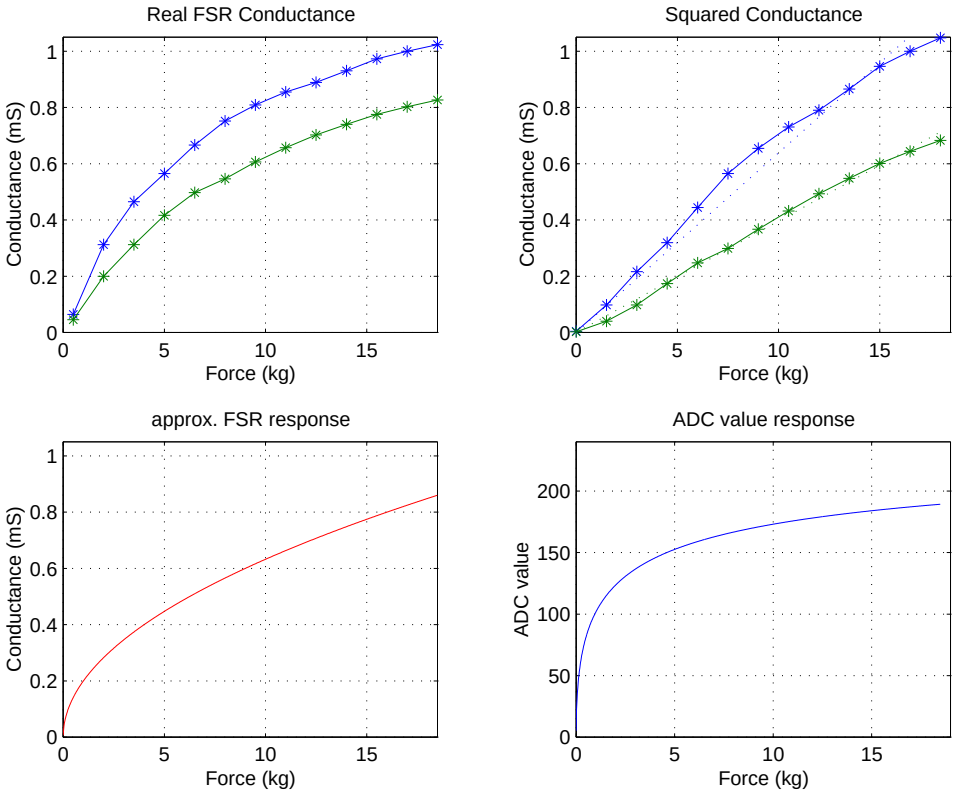


Figure 3: FSR response

The plot in the upper left of figure 3 shows the conductances of two FSRs versus the applied force. Since the IEE FSR technology datasheets[5] provide only a few sample points, an experiment was realized to measure the characteristic curve. The experiment is described in appendix A.3, which also contains a table with the measured values.

In the upper right plot the conductance was squared. The squared conductance curves are almost linear. The dashed lines are linear approximations. They suggest that the FSRs conductance depends linearly on the square root of the force. Therefore the following model is used:

$$g_{FSR}(F) = m\sqrt{F}$$

where m is approximately $200\mu S/\sqrt{kg}$[6].

When the conductances of two FSRs are compared to each other they can differ remarkably. The datasheet indicates a difference of up to $\pm 25\%$. Besides this, other effects as aging, destructive properties of adhesives etc. will play a role. What these differences look like is illustrated in the upper right graph in figure 3. Both straight lines go through the origin, but have different slopes. Therefore the factor *gain* is

[5] see `http://www.iee.lu/fsr1.htm`

[6] $m = g/\sqrt{F} = 0.8\mu S/\sqrt{15kg}$

introduced to model these differences. Now,

$$g_{FSR}(F) = gain \cdot m\sqrt{F} \tag{1}$$

The factor *gain* is a parameter that has to be estimated by comparing different FSR responses to a common force.

2.2 Floor Tile Output Model

Figure 4 shows a single floor tile. The cover is laid out on six supporting points. A force pressing against the center of the floor cover is distributed equally to the six supporting points. Three corners are equipped with load sensors which are connected in parallel, i.e. their conductances are added up to represent the total force on the three load sensors.

The conductance is measured indirectly by using a voltage divider, as depicted in figure 5. The voltage V_{out} over the resistance R is converted to a digital value ranging from $0 \ldots 255$ by an 8 bit analog digital converter. This digital value will be refered to as ADC value (analog digital converted) or simply called floor tile output.

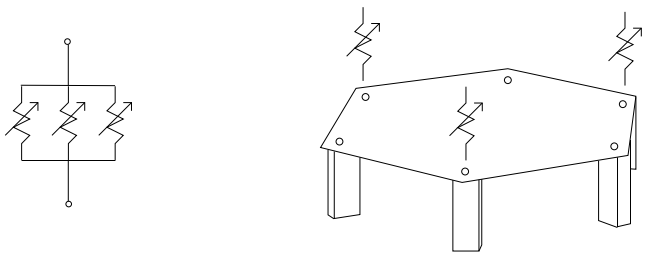


Figure 4: Floor tile of the 4x4 prototype floor. There are three force resistive sensors in parallel. The conductance of the floor tile g_{FSR} is the sum of these conductances. Additionally to the force that a person applies to a sensor, the sensor is preloaded by the weight of the floor cover. But since the preload is usually small (in the range of a few kg) compared to F (5kg to 120kg) it is neglected here.

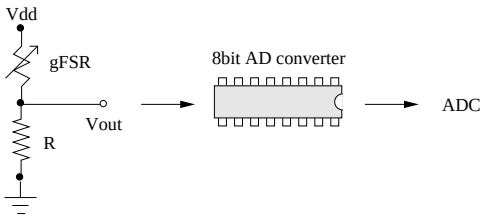


Figure 5: Voltage divider set up

From figure 5 the voltage divider setup gives

$$ADC(g_{FSR}) = rs\frac{g_{FSR} \cdot R}{g_{FSR} \cdot R + 1} \tag{2}$$

where rs is the full scale output of the analog digital converter. The ADC value is what we get from the hardware. Once the ADC value is known, the conductance

g_{FSR} can be computed by solving eq. 2 for g_{FSR}:

$$g_{FSR}(ADC) = \frac{ADC}{R(rs - ADC)} \tag{3}$$

Solving eq. 1 for the force F, we get

$$F(g_{FSR}) = \left(\frac{g_{FSR}}{gain \cdot m}\right)^2 \tag{4}$$

3 The γ-conversion

3.1 Introduction

When a force F is applied on a sensor a, we have to expect a different conductance than when we would take sensor b, because of inequalities among sensors in manufacturing, age, mechanical condition and so on (see figure 2). In the model this can be expressed by assigning different values to $gain_a$ and $gain_b$. We would like to know the values of these gains, since this would allow us to give a precise estimation of a persons weight.

But this requires knowing the force F corresponding to a measured ADC value, or at least having a good estimation for F. But that's exactly what we don't have, since the challenge is not to use any known test weight! Absolute values of F are not available.

However, we are not interested in a calibration which yields to a precise, absolute weight estimation, but compensates the sensor differences. If we know that a sensor is somewhat weaker or stronger than the average of all the others, the gain value should be chosen accordingly.

3.2 The basic Idea

Imagine you are walking in a group of persons. Every person has its own prefered speed, some like walking faster, some like it slower. The group has to agree on a common speed to prevent it from splitting up. What you in principle will do is that you compare your own speed with the average speed of the other group members. If you are slower, you will accelerate, if you are too fast, you will slow down. Note that to do so you neither need to know your absolute speed, say, in km/h, nor the absolute speed of anyone else. It is sufficient to compare the relative speeds.

This idea is now applied to our calibration problem. Adapting your walking speed corresponds to calibrating a sensor. Instead of estimating the gain of a single sensor (estimating your absolute speed), the ratio of gains from pairs of sensors will be estimated (comparing relative speeds). This ratio will be called γ. It measures the gain of a sensor in relation to the gain of another sensor.

Now the output of a tile can be calibrated as follows: based on the local output and comparisons between the output of the local tile and outputs from foreign tiles, we try to predict the average output of all the other tiles. This prediction can be used as the calibrated output, because all tiles try to mimic the behaviour of an average floor tile.

If we know the value from sensor a, and we also know the ratio γ, which will be defined shortly as $gain_a/gain_b$, we can predict the value of sensor b. This can be generalized if sensor b is not a specific physical instance of a sensor, but an artificial substitute which stands for all other floor tiles.

3.3 The definition of γ_{ab}

γ_{ab} is defined for a pair of sensors a and b as follows:

$$\gamma_{ab} = \frac{gain_a}{gain_b} \tag{5}$$

The point of view is sensor a. Sensor a is supposed to belong to the tile we want to calibrate (the local one), and sensor b is the tile we use as a foreign reference. γ_{ab} is therefore the gain of the local tile relative to the gain of a foreign tile. It expresses how strong the output of the local tile is relative to a foreign tile.

3.4 How to compute γ_{ab}

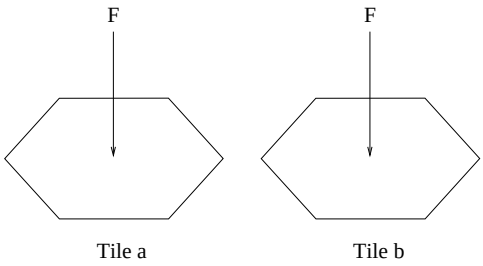


Figure 6: Comparison of two tiles

In the experiment depicted in figure 6 the force F is applied to two different sensors by a person stepping from tile a to tile b. Knowing that $F_a = F_b$ and using eq. 4, we get

$$F = F_a = F_b = \left(\frac{g_{FSR_a}}{gain_a \cdot m}\right)^2 = \left(\frac{g_{FSR_b}}{gain_b \cdot m}\right)^2$$

which can be reduced to show

$$\frac{gain_a}{gain_b} = \frac{g_{FSR_a}}{g_{FSR_b}} \tag{6}$$

If a is the ADC value from sensor a, and b the ADC value from sensor b, and we combine eq. 3, 5 and 6 we can compute γ based on the ADC values from the two tiles: $\gamma_{ab} = \frac{gain_a}{gain_b} = \frac{g_{FSR_a}}{g_{FSR_b}} = \frac{a}{R(rs-a)} / \frac{b}{R(rs-b)}$

$$\gamma_{ab} = \frac{a(rs-b)}{b(rs-a)} \tag{7}$$

3.5 γ_{ab}-values and their meaning

The definition of γ_{ab} allows to compare all the floor tiles pairwise. For each comparison the value of γ_{ab} has the following meaning:

$\gamma_{ab} < 1$: sensor a has a weaker output than sensor b
$\gamma_{ab} = 1$: both sensors are equal
$\gamma_{ab} > 1$: sensor a has a stronger output than sensor b

3.6 The γ-conversion

From the experiment depicted in figure 6 we know that the output of sensor a and of sensor b can be expressed without using the force F. This is the statement of equation 7.

Using eq. 7 it is now possible to compute b if a and γ are known[7], which will further be referred to as γ-*conversion*:

$$b = \frac{a \cdot rs}{\gamma rs - \gamma a + a} \tag{8}$$

Example. Figure 7 shows a plot of a γ-conversion. The local view corresponds to the abscissa, which belongs to sensor a, whereas the ordinate is the output of sensor b. The pair of values that was read from the sensors for a common force was 128 for sensor a and 192 for sensor b. According to eq. 7 $\gamma_{ab} = \frac{128 \cdot (256-192)}{192 \cdot (256-128)} = 1/3$. Using this curve it is possible to predict the value of sensor b if the value of sensor a is known, e.g. the value 64 reported by sensor a corresponds to the value 128 reported by sensor b. The γ-curve connects all the pairs (a, b) that share a common force.

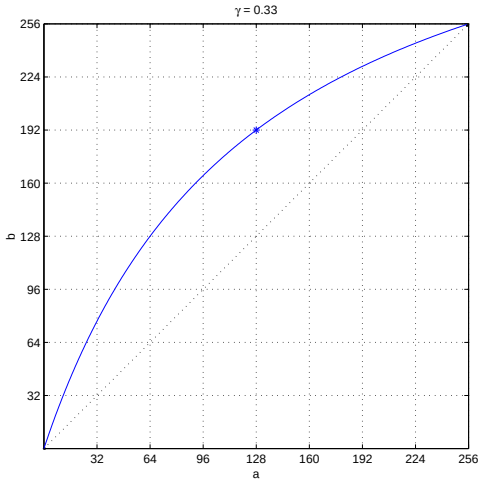


Figure 7: Plot of a γ_{ab}-conversion for the pair (128,192). The local view corresponds to the abscissa (sensor a), whereas the ordinate is the output of sensor b. The γ-curve connects all the pairs (a, b) that share a common force.

3.7 Comparing pairs of tiles

The principle to use the γ-conversion is now generalized. Instead of having only a single γ_{ab}, a tile is compared with several other tiles for different common forces. To get a good estimation of γ we should compare as many (a, b) pairs as possible. γ should then be chosen such that it minimizes the mean γ-*conversion* error, e.g. minimizes the error in the least squares sense:

$$E = \sum_i \left(\frac{a_i \cdot rs}{\gamma rs - \gamma a_i + a_i} - b_i \right)^2 \quad \text{min.}$$

[7] the subscript ab is omitted because the γ-conversion can be used as well as for γ_{ab} and for $\hat{\gamma}$, which will be introduced later

which means finding the root of the first derivative of the previous equation:

$$\frac{\partial}{\partial\gamma}E = 2rs\sum_i \frac{a_i(rs-a_i)(a_ib_i - a_ib_i\gamma + rsb_i\gamma - a_irs)}{(a_i+\gamma rs-\gamma a_i)^3} = 0$$

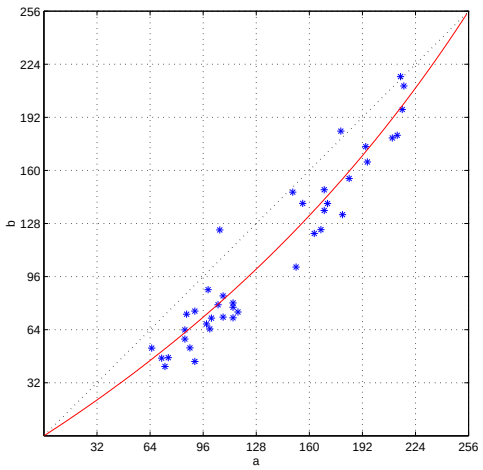


Figure 8: Measured (a, b) pairs are represented by asterisks, the solid line is the $\hat{\gamma}$-curve that tries to minimize the conversion error.

Instead of doing so we use an approximation which is a much simpler yet less resource consuming method. For each pair (a_i, b_i) $\gamma_{a_ib_i}$ is computed. It is reasonable to take some kind of mean value as γ. Since γ is a ratio, the harmonic mean is the appropriate choice[8]:

$$\hat{\gamma} = \frac{N}{\frac{1}{\gamma_1} + \frac{1}{\gamma_2} + \ldots + \frac{1}{\gamma_N}} \tag{9}$$

As a couple of numerical examples showed, the harmonic mean solution is close enough to the least squares solution. In the example given in figure 8, γ is:

no calibration:	$\gamma = 1.0$	$E =$	78166
arithmetic mean:	$\bar{\gamma} = 1.5655$	$E =$	5787
harmonic mean:	$\hat{\gamma} = 1.4571$	$E =$	5541
least squares:	$\gamma = 1.5095$	$E <$	1

Numerically computing the least squares solution would require storing all the (a, b) pairs, thus, the more calibration pairs are available, the longer finding the minimum will take. Since the calibration process is supposed to continuously run for several weeks, a huge number of pairs would have to be considered. This is not feasible. In contrast, $\hat{\gamma}$ can be computed iteratively, only two variables are needed: one to hold a running sum, another to hold the number of terms.

[8]Let us have a look at an example: Assume you compare your salary to the salaries of your friends. For the sake of simplicity we use normalized values, e.g. your salary equals one, your first friend earns twice as much as you do, and second one gets the fourfold of yours. So the average of your friends salaries (the arithmetic mean) is $(2+4)/2 = 3$. Now let us do the same again, but this time we use the ratio of your salary and a friends salary. So you earn $1/2$ of the first friend, and $1/4$ of the second one. If we would use the arithmetic mean too, we would get $(1/2+1/4)/2 = 3/8$ and not $1/3$. Instead, we have have to use the harmonic mean to get $2/(1/2+1/4) = 1/3$, as we would expect it from our previous consideration.

3.8 Evidence for correctness

To get an evidence for the correctness of the suggested model two real sensors are compared. Table 1 in appendix A.3 contains the values of the conductances for three different FSRs. There are thirteen measurement points for forces in the range of $0.5kg \ldots 18.5kg$. The experiment to find these values is described in appendix A.3.

The conductance is converted to an digital value using the voltage divider formula (eq. 2). Two out of the six measurements are picked from the table. Both sensors have their own axis in the graph depicted in figure 9. There is an asterisk for each measurement force.

The harmonic mean fit for the $\hat{\gamma}$-curve is also shown. Although the curve does not fit perfectly it suggests that the model is fairly accurate.

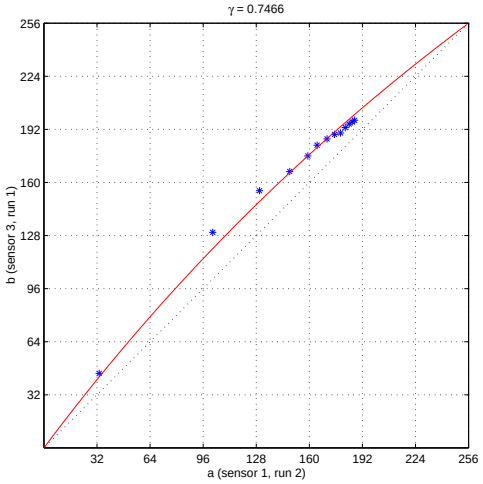


Figure 9: Justifying the model by real measured sensor characteristics.

4 Footstep Recognition

4.1 Introduction

The following section deals with the question how a step from one tile to another can be recognized. It is important that steps which are suited for calibration can be distinguished from ones that are not. For example, steps must be unambigous when people are crossing on neighbored floor tiles or even stepping together on a single tile. It's very likely that persons step on an edge between two tiles, or even on a junction of three tiles.

4.2 Getting the Steps

Figure 10 demonstrates how the steps can be extracted from the raw ADC signal. First, the signal is highpass filtered. The time constant is about $30s$. This removes the DC offset which arises mainly due to the preload of the floor tiles by the weight of the plate laid out on the sensors. To classify the filtered value into two classes a threshold is used. This value is very noncritical, even with small children the steps can be clearly distinguished from DC.

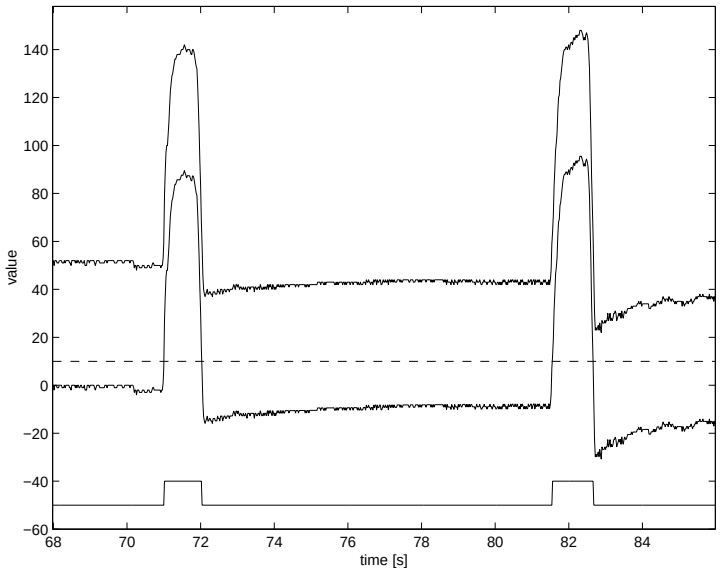


Figure 10: Extracting steps: The curve on the top shows the raw ADC value. By applying a highpass filter we get rid of the DC offset (curve in the middle). This makes it possible to use a threshold (dashed line) to find the steps. The extracted steps are plotted on the bottom.

4.3 Representing a step by a single parameter

Figure 11 depicts the time course of the force that was measured by a floor sensor for a sample step. The step was extracted from real data. The shape of the force is influenced amongst others by the walking style, the walking speed, the type and the size of the shoe and the spot where the foot is put down on the hexagonal floor tile.

The maximal force that a person can apply on a floor tile is not limited by its body weight. The faster a person moves, the harder the heel will hit the ground. Tiptoing will result in rather smooth force curves whereas marching or even stamping will produce strong peaks.

The calibration is based on comparing pairs of forces. The force on a floor tile is not constant during a step. Therefore, a single value representing the force on a floor tile has to be determined from the force time course. This value should be highly related to the persons weight and be as independent as possible from all the properties of the walking style.

Three possibilities are considered here: mean, maximum and median. The mean has the disadvantage that it depends on the proportion of the time that is used to step on/off a tile and the time the tile is stood on. Usually, the value is smaller than one would intuitively expect. It is therefore not considered here.

The maximum is stable as long as the walking style is constant. The first implementation of the calibration was based on this method. The result was astonishingly good. But it depends heavily on the way a person brings the force to the floor tile; tiptoing will differ remarkably from stamping.

The most stable solution, which is used for the calibration now, is to take the median value[9]. Figure 11 shows a typical footstep, the horizontal line corresponds

[9]Note that as long as a person sticks to a specific walking style, all the three suggested methods work well.

to the median. The drawback of the median is that it is computationally more expensive than the others.

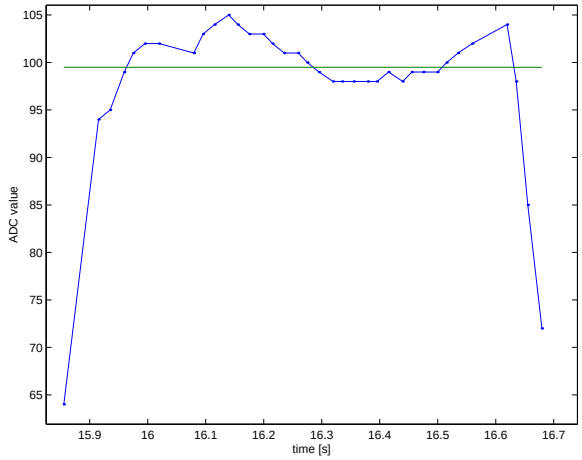


Figure 11: Taking the median as a stable property of a step.

4.4 Gauging steps

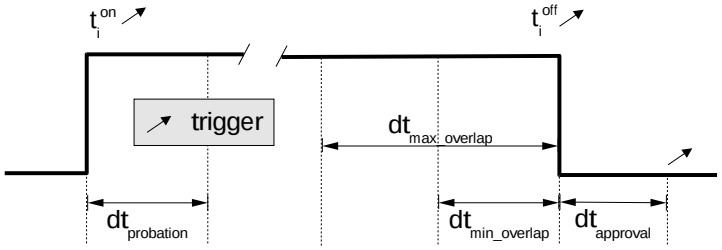


Figure 12: Gauging a step

Besides the force, the time of a step can also be measured. The next paragraph defines important timing terms which will be used to detect moves. Figure 12 illustrates how a step can be gauged.

t_i^{on} is the time at which a step begins on tile i (i.e. the adc-value crosses the threshold from below). Analogously t_i^{off} is the time at which a step on tile i ends (i.e. the adc-value crosses the threshold from above). A step has a period of probation, called $dt_{probation}$, which has to be passed in order to consider t_i^{on} a valid start time. This eliminates steps that are too short or are evoked by noise. A step from one tile to another (further called *move*) will be recognised by the overlap of $dt_{overlap} = t_j^{on}$-t_i^{off}. $dt_{min\_overlap}$ is the minimal required overlap for a move, $dt_{max\_overlap}$ the upper overlap limit. After the source tile has been left, the person must stay at least during $dt_{approval}$ on the destination tile.

Figure 12 and figure 13 illustrate how these terms are justified. The goal is to define the conditions that have to be met that a walking person can be tracked.The right leg is drawn bold. The step on tile A begins in phase *Ia)* as soon as the right foot is put on the floor. The step ends between phase *II)* and *IIIa)* once the right leg is removed from the floor.

- Before the right foot can be raised, the left leg has to be moved forwards to the next tile. This needs a certain time to complete. First, the full body weight has to be shifted to the right leg, second the left leg has to be moved forwards, third the full weight has to be put on the left leg to raise the right one. Therefore, we require a step to take longer than the period $dt_{probation}$.

- During phase *II)* the person is standing on two tiles concurrently. The steps in the step diagram overlap. The weight is shifted from the right leg towards the left leg at the speed the person is moving its body forwards. If the overlap is very big, the person is not really moving and the body weight pressing on the floor is distributed on two different tiles. Thus, the median value that is used to describe the force on a floor tile will drop and negatively influence the calibration process. Additionally the moment where the person moves from a tile to another can not be located in time precisely. The conclusion is that the overlap must be restricted to specific time interval, here called $dt_{max\_overlap}$. On the other hand, the overlap is the only feature that allows us to track people. If the overlap is too short, it is not easy to relate the two steps to each other. Therefore the overlap must exceed a minimum period, called $dt_{min\_overlap}$.

- The overlap in phase *II)* suggests that the person moves from tile A to tile B. If this really is the case, the person must stay on the destination tile for a certain amount of time, called $dt_{approval}$. The consideration is the same as for $dt_{probation}$.

The values for the introduced time intervals were determined empirically by judging the data collected from different persons (adults and children) walking or playing on the protoype floor:

$$\begin{aligned} dt_{probation} &= 200ms \\ dt_{min\_overlap} &= 50ms \\ dt_{max\_overlap} &= 400ms \\ dt_{approval} &= 200ms \end{aligned}$$

4.5 Conditions for a move

This section explains how people can be tracked using the step timing definitions given in the previous section. The goal is to identify moves from a source tile to a neighbored destination tile. This goal can be devided into two subgoals:

- In order to get a good calibration and weight estimation, we would like to collect as many moves as possible. The rules should not be too restrictive. We have to consider that in general it is not possible to unambigously track people by analyzing floor data. E.g. if they are close to each other they can swap their tiles, which can not be detected based on the thresholded floor data. Detecting moves would thus require that a person is always surrounded by a ring of non occupied tiles. But as we will see, if we assume that people do not clamber about in an artistic manner, some restrictions can be relaxed.

- On the other hand, if the amount of wrong detected moves is too big, the calibration will not work well. The timing intervals defined in section 4.4 have to be strictly abided.

This is an optimization problem. The rest of this chapter introduces a set of rules that allow to correctly track people most of the time. Have a look at figure 15 to get an idea about its capabilities. The algorithm was tested on real floor data of a 4x4 prototype floor. Within the outlined limitations it works very well.

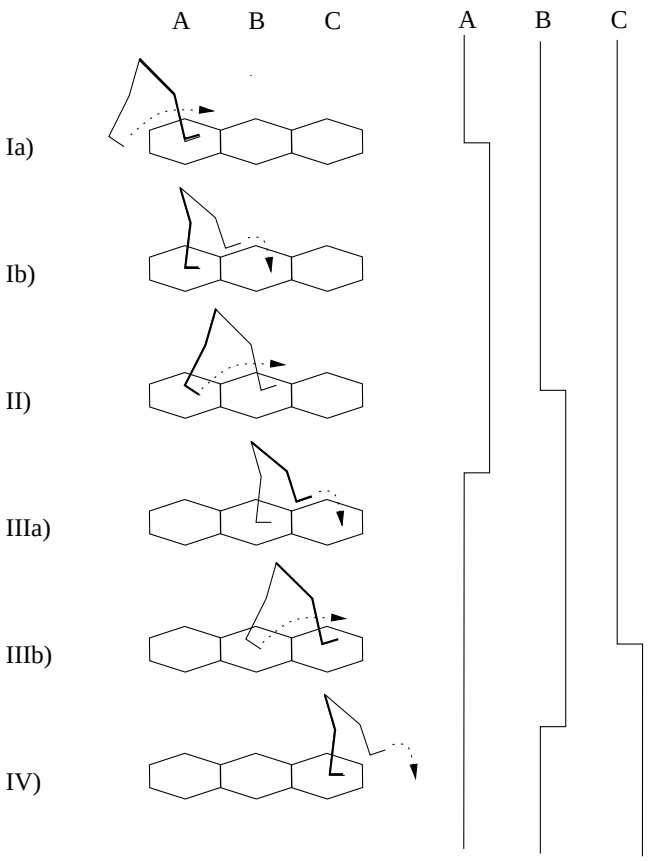


Figure 13: How moves can be extracted

The condition that a move from tile i to tile j has taken place is that (I) j is a neighbor of i, (II) the overlap time is within the valid range and (III) the move probated and approved.

Before the algorithm is presented, two definitions that will be handy are introduced. s and t are steps, $neighbor(s)$ is the set of tiles around tile corresponding to s, $s.onset$ is the time step s begins, $t.offset$ is the time step t ends etc.:

$$\begin{aligned}&\text{overlap}(s,t) = \\ &\quad (t \in \text{neighbor}(s)) \wedge (\text{s.onset} < \text{t.onset} < \text{s.offset}) \wedge (\text{t.offset} > \text{s.offset})\end{aligned}$$

$$\begin{aligned}&\text{strict\_overlap}(s,t) = \\ &\quad (t \in \text{neighbor}(s)) \wedge \\ &\quad (MAX(\text{s.onset} + dt_{probation}, \text{s.offset} - dt_{maxOverlap}) < \text{t.onset}) \wedge \\ &\quad (\text{t.onset} < \text{s.offset} - dt_{minOverlap}) \wedge \\ &\quad (\text{s.offset} + dt_{Approval})\end{aligned}$$

overlap(s,t) will be used to detect possible moves and to reject moves that are amiguous. strict_overlap(s,t) takes the time intervals defined in section 4.4 into account. It is used to verify moves.

Move detection algorithm. The starting situation is that somebody left a tile. Now the question is if it is possible to track the person, in other words, if it is

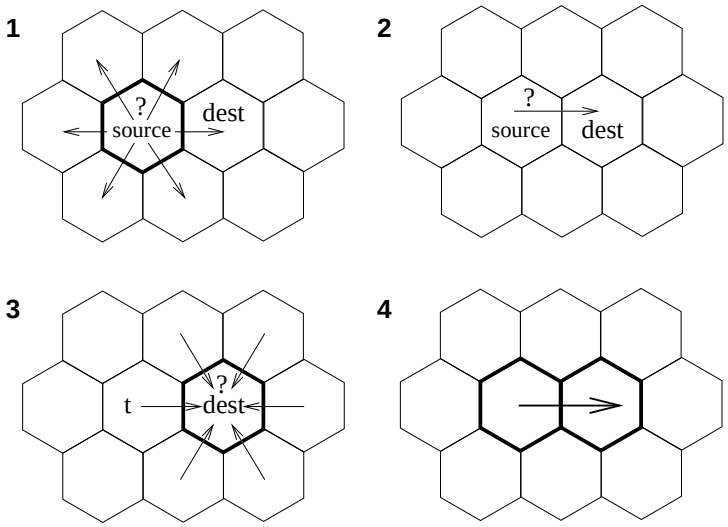


Figure 14: Illustration of the move detection algorithm. The tile named *source* is left. 1) All neighbored tiles are checked for overlaps. 2) If exactly one overlap is found, here with tile *dest*, it is verified to be a strict overlap. In the case this is true 3) all overlaps from tiles neighbored to *dest* are searched. If there is exactly one overlap, this must be with tile t, which in turn must be the same tile as *source*. Therefore, a unique step is found.

possible to detect a unique move to a neighboring tile. The tile being left is called *source*.

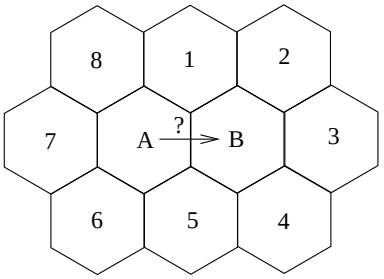
1. find all the steps *dest* on the six neighboring tiles that satisfy the condition overlap(*source*,*dest*). if more than one step or none is found, reject the move. else
2. verify that *dest* satisfies the condition strict_overlap(*source*,*dest*). if it turns out to be false, reject the move. else
3. find all the steps t on the six tiles neighbored to *dest* that satisfy the condition overlap(t,*dest*). if more than one step is found, reject the move[10]. else
4. verify that t satisfies the condition strict_overlap(t,*dest*). If the result is true[11] a unique move from *source* to *dest* is detected.

Figure 14 illustrates the functioning of the move detection algorithm.

4.6 Examples

Figure 15 gives some examples why certain moves are considered valid and others are not.

[10] at least one step must be found, namely *source*
[11] t must be the same as *source*



Labeling of floor tiles

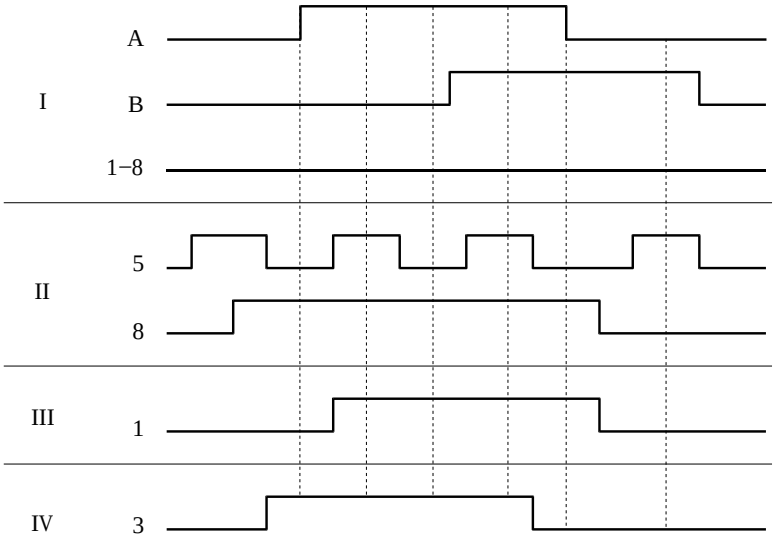


Figure 15: This figure illustrates how the given rules can be applied to detect a unique move from A's point of view. Someone who stands on tile A walks away, and we would like to know where he went by looking at the timings of the step diagram above.
I) This is a valid move from A to B. The overlap of A and B is within the required range. Since there is no step on all neighbor tiles of A and B, the only possibility to leave A is stepping onto B, and the only way to go to B is coming from A.
II) Despite the additional steps on tile 5 and 8 the move remains valid. The taps on tile 5 do not overlap with the end of the step on tile A, so there can't be a move from tile 5 to tile A. This is not the case for the step on tile 8, which ends after the one on tile A, but it starts before someone stepped onto tile A, so there can't be a move from A to 8. Therefore the move from A to B is still unique.
III) No unique move can be extracted. It is true that the step on tile 1 ends before the probation time and therefore a move from A to 1 would never be considered, but it remains longer than the step on tile A, so it is unclear which of the two possibilities is the correct one.
IV) The only way to leave A is to step on B, but there are two possibilities how someone might have got on tile B, namely tile A and tile 3. Therefore the move is invalid. This can happen if two persons move together on a single tile or the person on tile A jumps to another location.

5 Traces

5.1 Introduction

The previous section described how moves can be detected. A move is related to two tiles, namely *source* and *dest*. It states that a person first stood on tile *source*, then moved to tile *dest*, thus the output from the two tiles can be compared to each other, since they share a common force.

Consecutive moves can be put together to a trace, so a trace is nothing else than a set of steps.

Advantages. Introducing traces has two advantages, namely:

- Improving the γ - conversion
- Stabilize the running weight estimation

Each detected move gives one pair of tiles which can be compared, thus one additional γ_{ab} value per tile. If consecutive moves belonging to a single person are grouped by a trace, every pair of steps which is part of it can be used to compare two tiles. For a trace of the length n, $\binom{n}{2}$ pairs are possible. E.g. a trace that contains the last eight steps provides $\binom{8}{2} = 28$ comparable pairs! Compared to the seven pairs that can be extracted from single moves this is a tripling of calibration possibility.

Instead of taking the calibrated value from the current tile as basis for a weight estimation, the mean value along a trace can be used. This makes sense because by definition the trace only contains steps corresponding to a single person. Differences among steps caused by, e.g.

- sensor variabilty
- unequal distribution of the force to the sensors by not stepping on a tile's center
- different walking styles
- unprecise calibration in an early adaption state
- noise

can be averaged out.

5.2 An efficient way to get all step pairs

Figure 16 explains how all possible step pairs along a trace can be computed efficiently. If n is the number of steps along a trace, then the number of pairs is $\binom{n}{2}$=$n(n-1)/2 = O(n^2)$. If these pairs are computed incrementally as soon as a new step is added to the trace, the computational expense is linear with the number of steps. The lastest added step n is compared to all previous steps $1 \dots (n-1)$.

5.3 Pseudocode of Trace Algorithm

The following algorithm is used to determine the traces:

```
for(all tiles)
{
   if( current adc value > threshold )
```

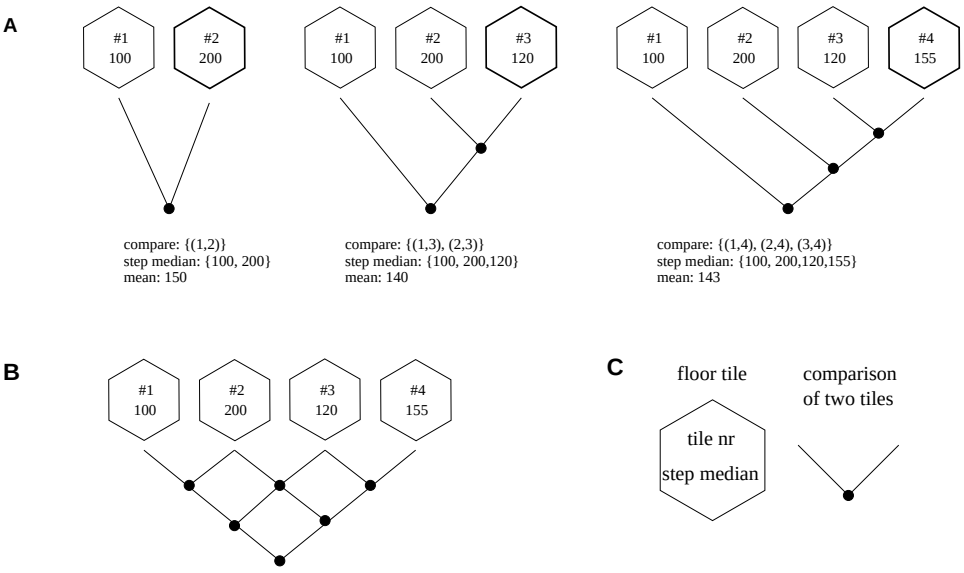


Figure 16: Example how step pairs can be computed efficiently with linear increase in time and how averaging can be used to stabilize a persons weight estimation. Have a look at *C)* to get the explanation of the symbols. *A*) The tile currently stepped on is marked with a thick border. This tile is compared to all previous tiles. *B)* Shows the result after stepping on all four tiles. All $\binom{4}{2} = 6$ pairs are compared.

```
{
   // someone stands on the tile
   if( the step flag of the current tile is not set )
   {
      // someone freshly stepped on
      create new step info data structure, store the step-on time and
                          add it to the step list of the current tile
      set the step flag on the current tile
   }
   else
   {
      // someone is still standing on the tile
      update the step value if necessary (step median)
   }
}
else
{
   // maybe someone left?
   if step flag set: store step-off time and unset step flag
}
// mark
for( go through the step list of the current tile )
{
   if( current step has reached due time )
   {
      if( the step has not yet a trace list )
      {
```

```
            create one and register it globally
         }
         add the step to the trace
         try to find a unique step
         if( a unique step is found )
         {
            pass the trace to the next step
         }
         else delete the trace
         mark the step for deletion
      }
   }
}

// sweep
for(all tiles)
{
   go through the list of step info data structures
   {
      remove all the marked ones
   }
}
```

6 Implementation

The ideas described in this report are implemented as a `C++` standalone program. Memory is allocated dynamically, the `list` class from the the standard template library (STL) is used as a container for dynamical data. A small tool to detect memory leaks was also written and added to the code. There are 1297 lines of code, compiled with `gcc`. A `Makefile` is also given, the code can be compiled by running `make`. The resulting executable has a size of 28kBytes.

6.1 Interface

```
Calibrator( const FloorGeometry *floorGeometry );
```

`Calibrator` is the constructor. It initializes its data structures. A pointer to a `FloorGeometry` object has to be passed. This is needed to know how many tiles there are and to find neighbors.

```
int process( struct timeval& TimeWhenDataAcquired,
             const int RawData[],
             int CalibratedOutput[], int StepOutput[] );
```

Processes raw ADC values. The `RawData[]` array must provide the ADC values measured at the time `TimeWhenDataAcquired` for the number of tiles that are passed by the `FloorGeometry` object. The valid range of ADC values is $0 \ldots 255$. It may also contain elements with -1, this is to indicate tiles which are at least temporarily broken and are not reporting any new data. The return value from `process()` is zero if no error occurs. Currently the return value is always zero, if memory can not be allocated, an exception will be thrown.

After the call `CalibratedOutput[]` contains the γ converted (i.e. calibrated) outputs in the range $0 \ldots 255$ or -1 if the no input value was provided. `StepOutput[]` contains 0 for each tile if there is no step or a positive value estimating the mean weight of the step along its trace.

```
int GetGammas( double Gamma[] ) const;
```

Fills the array `Gamma[]` with the γ factors. The passed array must be big enough to hold the gamma values for all floor tiles.

```
int loadCalibrationState( const char* filename );
int saveCalibrationState( const char* filename );
int resetCalibrationState();
```

These three methods can be called to load, save and reset the calibration state. The return value is 0 if everything went okay or `EINVAL` if the file given by `filename` could not be accessed.

6.2 UML Diagram

Figure 21 (appendix A.2) gives an overview of the calibrator classes. The model of notion is called UML (unified modelling language), which is a widespread tool to represent anything related to object oriented design.

The mentioned figure shows a so called class diagram. There is a box for every class. The class name is given in the top of the box, the methods are listed in the middle and the bottom part contains the methods. The lines between the classes mean that the connected classes are related to each other. You can think of it as an object *owns*, *uses* or *accesses* another one.

6.3 Performance

For testing purposes a huge floor data file was created by a multiple concatenation of real measurements. It contains 80'516'000 samples (which corresponds to about 18 days). The number of tiles is sixteen, there are from one to four persons walking on the floor simultaneously.

It takes 6 minutes on a linux 2.2.16 PII-350 machine to process the file, so 225'000 process() calls are performed per second. The size of the executable is less than 28kBytes and the memory usage was below 700kBytes.

7 Results

Figure 17 illustrates how the calibration affects the floor data. It shows the plot of values from three different floor tiles produced by a single person walking on a 4x4 floor. Interesting is the time course of calibration. Note that within 30 seconds the three tiles are equalized. Figure 18 is zoomed in to emphasize the effect of calibration.

The histograms from three people are depicted in figure 19, both raw and online calibrated data is shown. In the uncalibrated case the data can not be segmented at all, whereas the calibrated step data has a very prominent peak for each person.

8 Conclusion

8.1 Discussion of the results

The presented solution achieves already a reasonably good calibration after a few steps, is able to run online during normal operation without needing a maintainance mode or the need to introduce known test weights, and it has a low memory and CPU resource consumption.

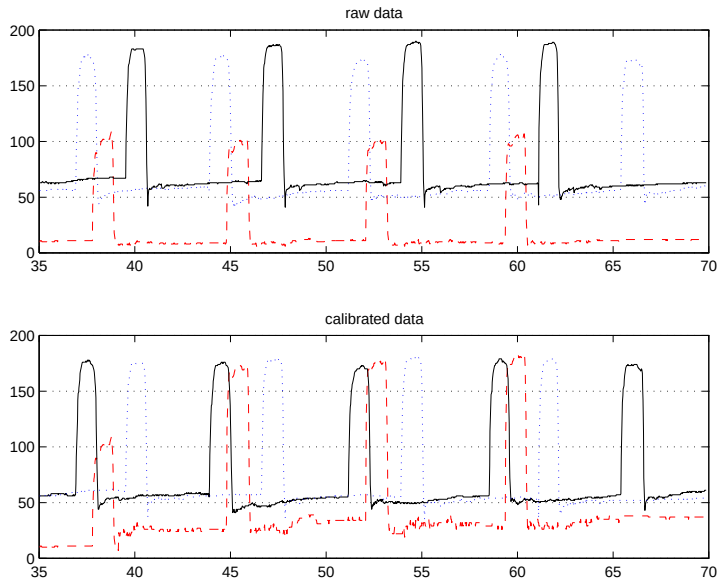


Figure 17: Online calibration of data from three different floor tiles and a single walking person: The upper graph shows the big differences in the raw data, while the lower one illustrates how the values equalize to each other.

8.2 Further considerations

Based on the calibrated step data the weight of a person can be estimated. Once the weight is approximately known, it can be related to the age of the person. A growth curve for boys from the age of 1 to 18 years is depicted in figure 24 (appendix A.4). Although this is very coarse it would allow to discriminate children from adults.

The trace information is currently used internally only. It could be exported to help other tracking systems based on vision or auditory devices to improve their performance. Going a step further, the tracking information from other systems could also help to improve the move detection algorithm.

Each of the floor tiles has three force resistive sensors. The calibration process treats them as a single one. Using the information of all three sensors, it might be possible to locate people with sub floor tile resolution. If the position on the floor tile could be estimated, this would also help to improve the calibration. It has to be mentioned though, that due to the hexagonal form of the floor tile the analysis of the force distribution across the six points of support is not trivial.

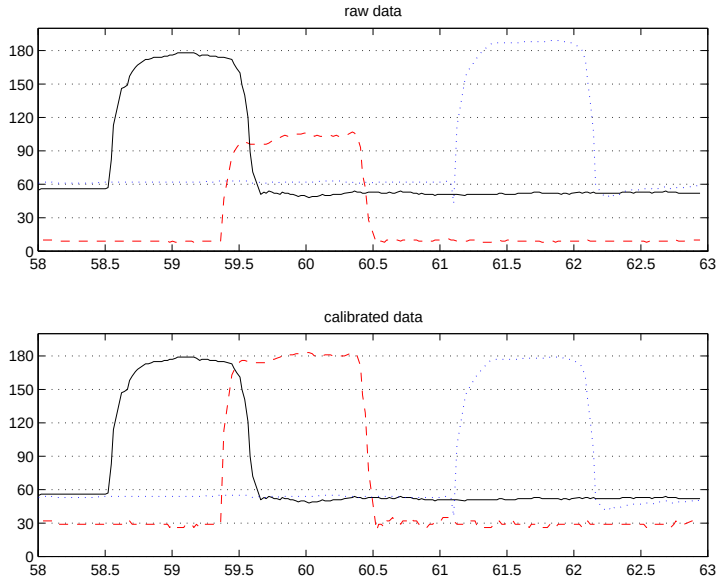


Figure 18: The same as figure 17 but zoomed in to better see how close the three tile values get after a short training time.

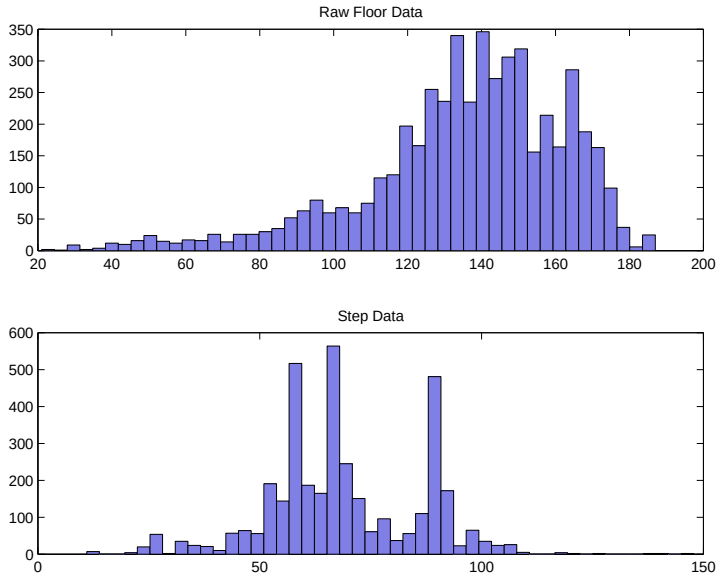


Figure 19: Histogram of extracted steps from three people with the weights 57 kg, 66 kg and 106 kg walking on a 4x4 prototype floor. The top histogram shows the raw floor data. Only values which belong to a step (i.e. values above step threshold) are considered. A segmentation is not possible. Lower histogram: These values are γ-converted and equalized along traces. There is a prominent peak for each person which allows to segment the data easily.

A Appendix

A.1 GUI

Footsteps, the calibrator GUI. The calibrator presented in this work is only a small part that is split off a much bigger standalone real time program that was developed since the beginning, called *footsteps*. Figure 20 shows a screenshot of the running program.

In contrast to the calibrator, *footsteps* has a GUI as a front end to the internal representation of the data structures. Its purpose was to

- have a tool to verify the models
- proove that the suggested algorithms work in real time
- detect systematic errors
- track problems more easily due to the graphical access to the internal state
- judge the performance on real data

Features. The program is able to read floor data directly from the serial port or from a log file that was recorded earlier. If the origin of the data is a log file, it can be replayed at different speeds.

The floor is represented graphically and can be rotated or mirrored to adapt it to the real orientation. Every floor tile is represented graphically and contains a graph with the pressure values of both raw and calibrated data of the last ten seconds and the corresponding γ-value. Stepped-on tiles are highlighted.

The following graphs are supported and are updated in real time:

- steps (diagram similar to digital circuits) based on thresholded values
- traces with all the steps and the mean pressure along the step
- (a,b) pairs used to compute $\hat{\gamma}$ and the corresponding γ-curve

The calibration state can be loaded from and save to a file.

Implementation. The object oriented program consists of 4300 lines of C++ code. It was developed using the integrated development environment `kdevelop`[12] under Linux. The toolkit QT[13], version 2.2.2, from Trolltech was used for the graphical user interface.

[12] `http://www.kdevelop.org`
[13] `http://www.trolltech.com/products/qt/qt.html`

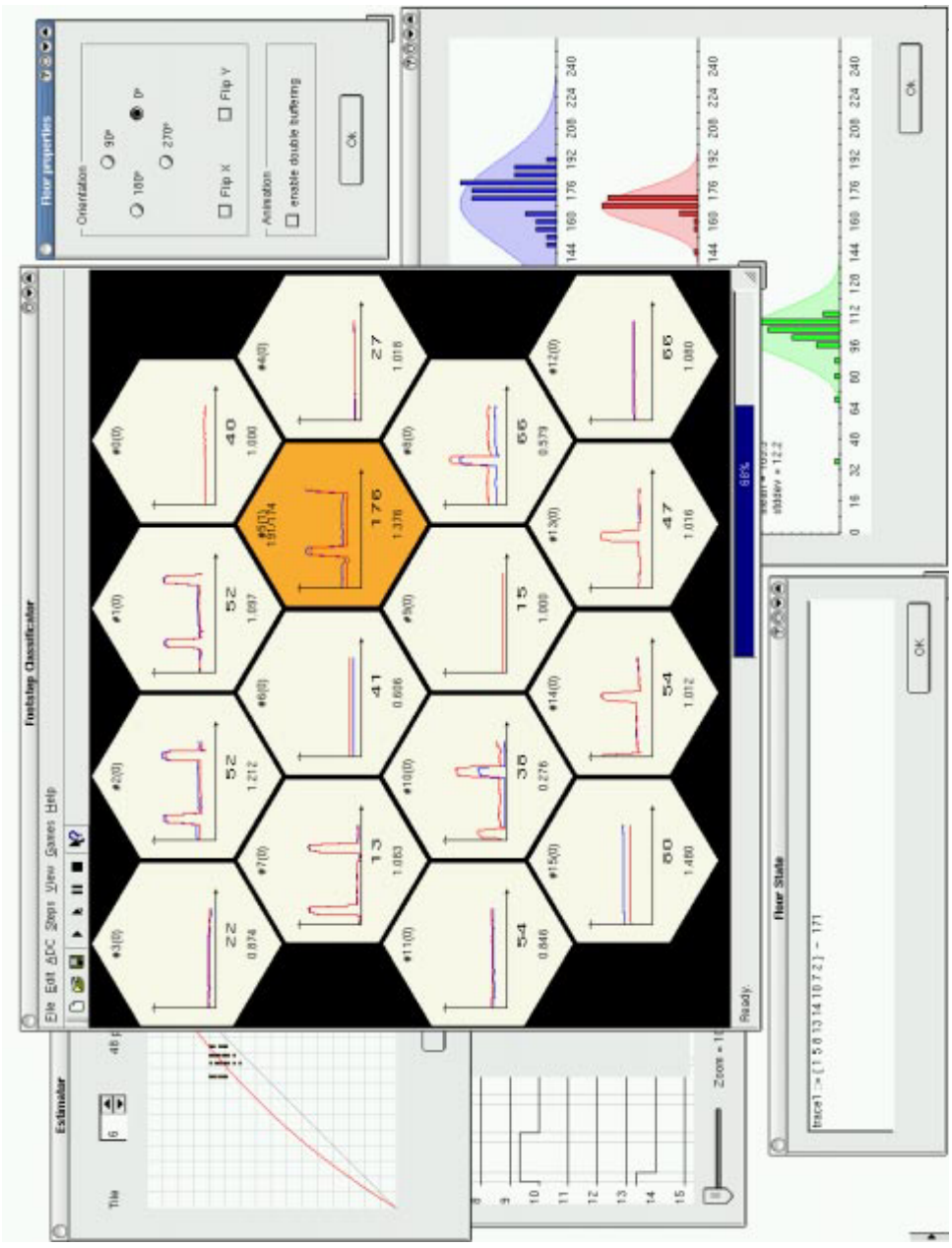


Figure 20: Screenshot of the calibrator GUI.

A.2 UML Diagram

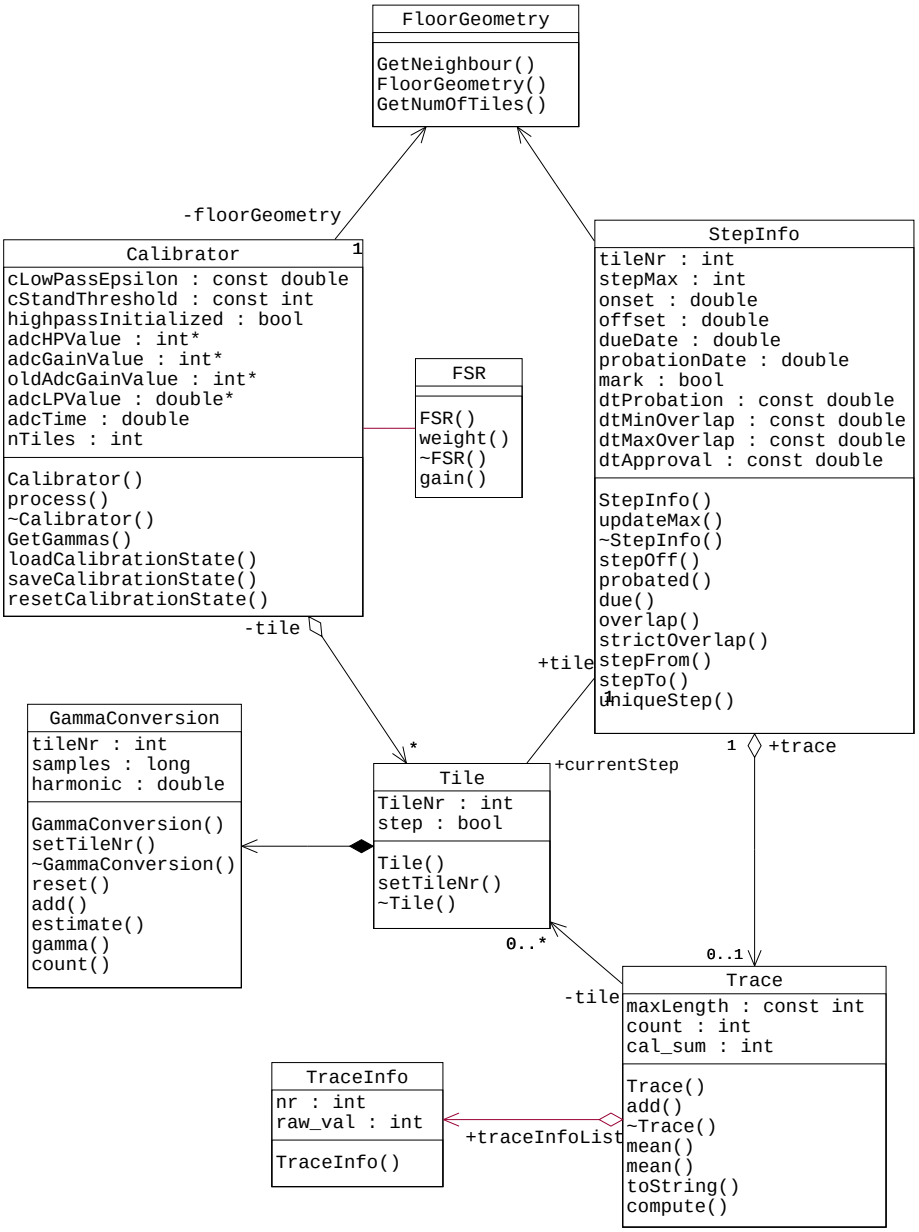


Figure 21: UML diagram of the calibrator classes.

A.3 FSR

The dependence of the conductance versus the force that compresses a force resistive sensor was measured for three FSRs. Every measurement was repeated twice. Figure 22 depicts the setup of the experiment.

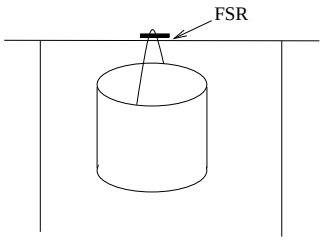


Figure 22: Experiment to measure the characteristic curve of an FSR. The FSR is put between a long narrow plank and a block. A bucket hung on the block. The weight of the bucket presses against the block which compresses the FSR. The resistance of the FSR is measured with a multimeter. The weight pressing on the FSR can be adjusted by filling the bucket with water.

The FSR was preloaded with about $0.5kg$ by the bucket and the block. The empty bucket was filled with water in steps of 1.5 liter until it contained 18 liters. The collected data is given in table 1, the plots are depicted in figure 23.

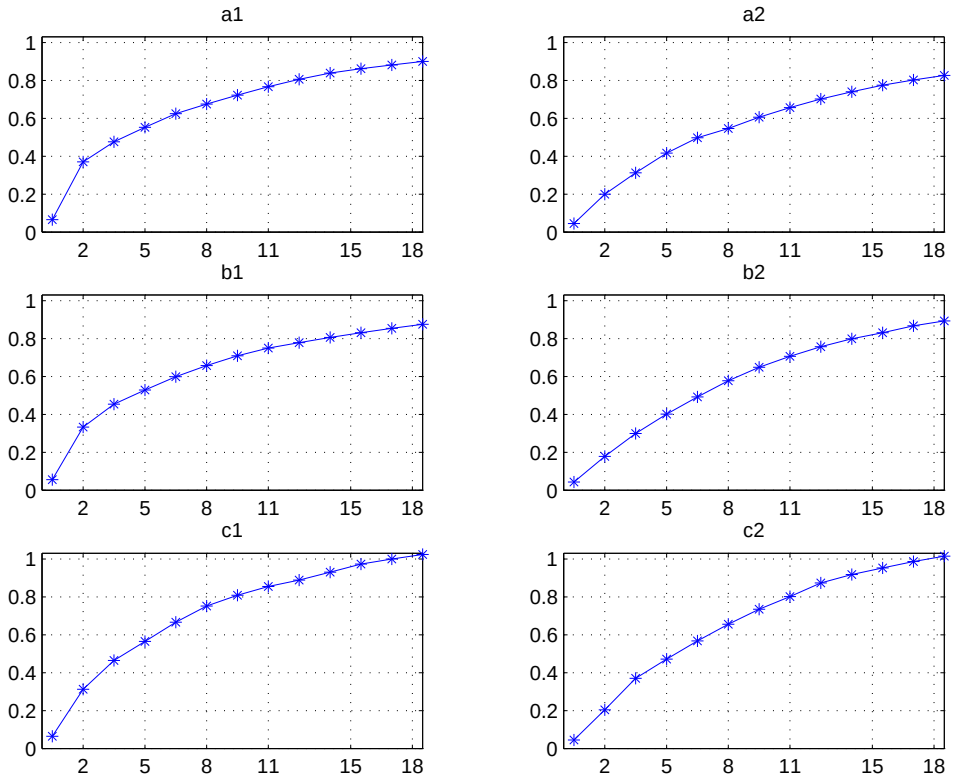


Figure 23: Graphical representation of table 1, force vs. conductance. The force is given in kg, the conductance is given in mS.

	Resistance (kOhm)					
Force(kg)	a1	a2	b1	b2	c1	c2
0.5	15	22	18	23	16	22
2.0	2.7	5.0	3.0	5.6	3.2	4.9
3.5	2.1	3.2	2.2	3.34	2.15	2.70
5.0	1.81	2.4	1.89	2.49	1.77	2.12
6.5	1.60	2.01	1.67	2.03	1.50	1.76
8.0	1.480	1.830	1.520	1.732	1.330	1.525
9.5	1.384	1.650	1.410	1.542	1.236	1.361
11.0	1.304	1.522	1.334	1.415	1.170	1.247
12.5	1.240	1.424	1.285	1.320	1.125	1.145
14.0	1.192	1.351	1.241	1.252	1.075	1.089
15.5	1.160	1.290	1.204	1.204	1.028	1.050
17.0	1.134	1.246	1.171	1.153	1.000	1.014
18.5	1.111	1.210	1.142	1.120	0.977	0.985

	Conductance (mS)					
Force(kg)	a1	a2	b1	b2	c1	c2
0.5	0.065	0.045	0.055	0.043	0.064	0.045
2.0	0.370	0.200	0.333	0.178	0.312	0.204
3.5	0.476	0.312	0.454	0.299	0.465	0.370
5.0	0.553	0.416	0.529	0.401	0.565	0.471
6.5	0.625	0.497	0.598	0.492	0.666	0.568
8.0	0.676	0.546	0.657	0.577	0.751	0.655
9.5	0.723	0.606	0.709	0.648	0.809	0.734
11.0	0.767	0.657	0.749	0.706	0.854	0.801
12.5	0.807	0.702	0.778	0.757	0.888	0.873
14.0	0.839	0.740	0.805	0.798	0.930	0.918
15.5	0.862	0.775	0.830	0.830	0.972	0.952
17.0	0.882	0.802	0.854	0.867	1.000	0.986
18.5	0.900	0.826	0.875	0.892	1.023	1.015

Table 1: Three force resistive sensors a,b,c, measured twice.

A.4 Growth curve

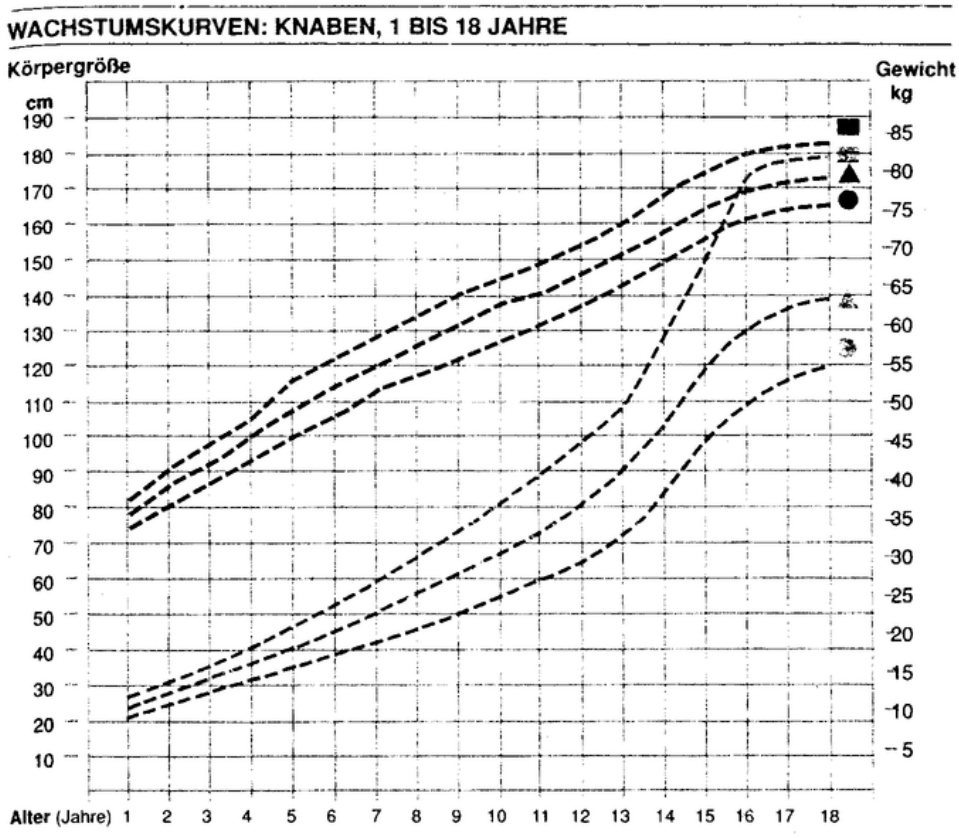


Figure 24: Growth curve for boys at the age of 1 to 18 years. The upper three curves correpond to the left axis which is the height in *cm*. The lower three curves correspond to the right axis which is the body weight in *kg*. Symbols: square means above average, triangle stands for average, circle denotes below average. (K.U. Benner, *Gesundheit und Medizin heute*, Bechtermünz Verlag, 1997)