reckonMe: Live collaborative pedestrian dead-reckoning on mobile phones leveraging audio-based proximity sensing

Diploma Thesis
Prof. Dr. Paul Lukowicz
Embedded Systems Lab
Department of Computer Sciences and Mathematics
University of Passau

written by
Benjamin Thiel

Second Examiner: Prof. Dr. Harald Kosch
Supervisor: Dr. Kamil Kloch

Benjamin Thiel
Matriculation Number: 46008
Grabengasse 9
94032 Passau
ben.thiel23@gmail.com

Abstract

This thesis presents *reckonMe*, an indoor positioning system that uses pedestrian dead reckoning (PDR) and leverages collaboration in order to improve its location estimates.

The system is implemented as a standalone iPhone app, which performs all calculations in real time, on the device. The phone is supposed to be placed in the trouser pocket while walking, where it tries to detect the user's steps and their direction, thereby computing his position. Furthermore, *reckonMe* makes use of inaudible sound emissions in the 18 kHz range to detect the proximity of two phones. Once two devices are detected as proximate, their estimates are exchanged via Bluetooth.

On average, doing this leads to an improvement of the position estimates, as it reduces the linearly growing error by 27.5 %. The presented approach is completely independent of infrastructure and low-cost as it uses only commodity hardware like an iPhone 4.

Contents

List of Figures

List of Tables

1. Introduction

With smartphones becoming ubiquitous, the vision of pervasive computing is closer than ever. Users are accustomed to having the Internet literally at their fingertips and to be guided by navigation systems around the world.

Although satellite based positioning systems like GPS are working very well outdoors, they struggle indoors, as their signal rapidly deteriorates indoors. A mass deployment of indoor positioning solutions has yet to occur. With the advent of smartphones however, indoor navigation gained new traction and for the first time, has a reasonable chance for widespread adoption.

Most publications either deal with specialized hardware (e. g. foot-mounted sensors) or require appropriate infrastructure to be present (e. g. radio beacons). Few publications try to leverage commodity off-the-shelf (COTS) hardware like smartphones for this task. The reasons for this include the challenging environment in which smartphones are typically carried: the trouser pocket. Neither are the movements of the subject's legs within the pocket very intense, nor very precise (due to movements of the phone in the pocket). However, a collaborative approach like reckonMe can cope quite well with these problems, as will be shown.

Indoor navigation systems in general, can be divided into two groups: beacon based and incremental positioning techniques. Beacon based systems supply absolute positions by calculating the relative position to known locations. Beacons are present at these locations, which continuously distribute signals using different mediums like radio, light or sound. The position relative to these beacons on the receiving side can be derived using different features of the signal, like its time of arrival (ToA), angle of arrival (AoA), the received signal strength indication (RSSI) or a combination thereof.

While these systems are often deployed outdoors, with GPS being a well-known example, they can also be deployed indoors. Examples are Landmarc [7], which makes use of cheap RFID tags, or Cricket [10], which uses ultrasound. Newer approaches often focus on exploiting signals that are already present, like WiFi signals, and use fingerprinting techniques to statistically determine the position using pre-recorded maps, like in [6].

Incremental location systems, like reckonMe, on the other hand, are inherently infrastructureless. Instead, they require an initial position fix (e. g. originating from GPS) from which the displacement is continuously calculated, thereby introducing a cumulating error. These systems can by summarized by the term dead-reckoning (DR), which has its roots in nautics and aeronautics. A DR system measures velocity and directional information and integrates it over time.

A pedestrian dead reckoning system (PDR) applies this principle to pedestrians. As it is hard to measure a pedestrian's velocity directly, it is typically approximated by detecting the pedestrian's steps. Frequently this is done using designated hardware like inertial measurement units (IMU), mounted to the foot for example, like shown in [8]. An IMU is an electronic device that measures the velocity, orientation and gravitational forces using state-of-the-art gyroscopes, accelerometers and sometimes magnetometers. The advantage of using an IMU lies in its high precision with known error characteristics, its disadvantage is the high cost.

As Steinhoff et al. show in [14], placing an IMU into the trouser pocket yields only slightly worse results than placing it on the foot. This is a promising result for using smartphones as a low-cost replacement for IMUs, although they are less precise. Jin et. al [3] tackle this problem by equipping the pedestrian with two phones, which perform a sensor fusion to greatly improve long-term accuracy.

In order to compensate for the ever growing estimation error of pedestrian dead-reckoning systems, Kloch et al. [4] have shown that the potentially unbounded error could be bounded to a constant range of approximately 25 m if many users were allowed to "collaborate". In this case, collaboration means exchanging each other's position estimate when two people and their devices are proximate to each other, thusly improving their estimates. The reason for this is an emergent behavior of collaborative localization systems, shown in [5], where even though the individual experiences a sizable error, on the large scale a "location awareness" emerges.

While [4] used the GPS ground truth (available due to outdoor experiments) to determine proximity in an offline evaluation, *reckonMe* tries to perform these tasks live, on the device. For this purpose, a live dead-reckoning combined with infrastructure-less audio-based proximity sensing and spontaneous exchanges using Bluetooth was implemented.

At first, the idea was to simply use Bluetooth visibility as a proxy for the proximity of two persons. However, experiments revealed that Bluetooth connections initiated by the iPhone API had a surprisingly high operational

range, making it unsuitable for proximity detection. Two persons carrying an iPhone 4 in their pants pocket and walking around in an office building, resulted in a visibility of up to 60 meters, even through several walls. With the persons facing each other in a shopping mall, the connection was operational within a range of up to 80 meters, the longest possible distance to be found in that mall. A similar test outdoors where the two persons' backs were facing each other while approaching, an operational connection could be established at a distance of 25 meters, still a long distance considering the fact that the radio waves were shielded by two people. Given that the iPhone is only specified as a Class 2 Bluetooth device, which specifies a maximum range of 10 meters, it performs exceptionally well. On the other hand, its WiFi and Bluetooth antenna lies on the outside of the device, which constitutes the frame of the phone.

In need for a finer-grained proximity detection, which must not require additional sensors or external infrastructure, an audio-based approach has been developed. While many methods have existed for proximity detection in general, there are only two possibilities that fulfill the above criteria. One is ambient sound analysis as proposed by [16]. This has the disadvantage that it requires appropriate sound to be present, which may not always be the case.

To compensate for this potential lack, the author chose to let the devices emit sound themselves through their speakers. In order to avoid disturbing users or interfering with other sound sources, an inaudible frequency band ranging from 18 kHz to 20 kHz has been chosen for this purpose.

The method consists of two steps. First, Bluetooth pairing is used to confirm that the two devices are broadly in the same space i.e., are "eligible" for the close-proximity test. Next, one device starts emitting repeating sound patterns while the other tries to detect them. This method works well in a variety of environments ranging from an empty office space to a busy mall. Upon successful detection of the emitting phone by the detecting phone, the two phones exchange their position estimates, with the intention of reducing their errors.

Localization using sound, as used in Cricket[10], has been proposed for several use cases, though none of them is applicable for the intended in-pocket ad-hoc usage in a smartphone-based PDR system.

Scott et al. [13] propose a system where microphones are pre-installed in a room and the system detects and locates user-generated sounds (i.e. clapping). In [16], Wirz et al. used fingerprinting to verify the existence of a relation between the distance of two devices and the similarity of the recorded ambient

sound. Girod et al. present ENSBox, a custom-built platform for rapidly deploying self-calibrating distributed localisation using acoustics in [2].

Arentz et al. [1] demonstrate that near-ultrasonic sound processing is feasible on iPhones and can even be used for data transfer in the short range. In [9], Peng et al. implemented an acoustic ranging system using smartphones yielding a high accuracy (up to 2 cm in the 10 meter range), though the phones were in line of sight. Acoustic localization using smartphones is even possible in 3D space, as shown by Qiu et al. in [11], using two microphones per phone in an unobstructed low-noise environment.

However, most of these approaches are hardly applicable to the case where both phones are carried in the persons' pockets, as the noise caused by the phone rubbing against the fabric during movement is dwarfing most signals. Instead of trying to measure the distance, this approach focuses on detecting whether two persons are standing proximate to each other or encountering each other through walking by.

2. Design

As the error in a PDR system constantly increases with the distance walked, a means for correction is desired. In the following, the system design of the iOS app *reckonMe* is presented, which performs PDR live on the device and leverages peer-to-peer exchanges in order to improve the position estimates. It is worth noting, that this approach is completely infrastructureless and works "out-of-the-box". The flowchart shown in figure 2.1 gives an overview of the process taking place while running the app.

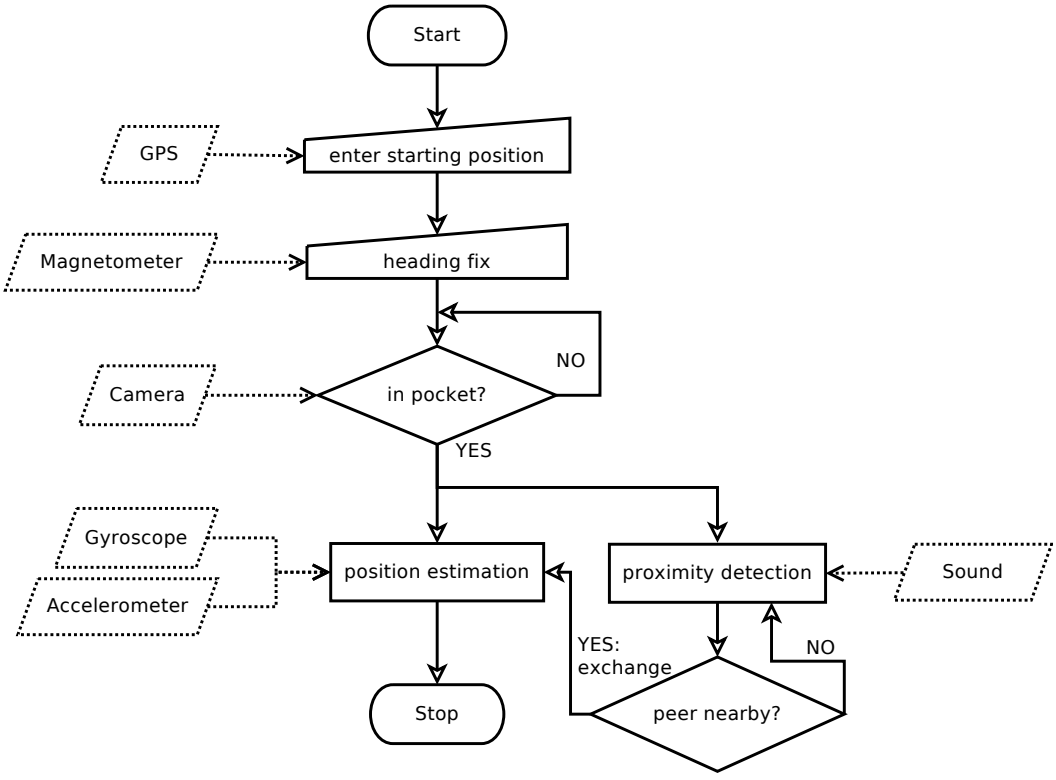
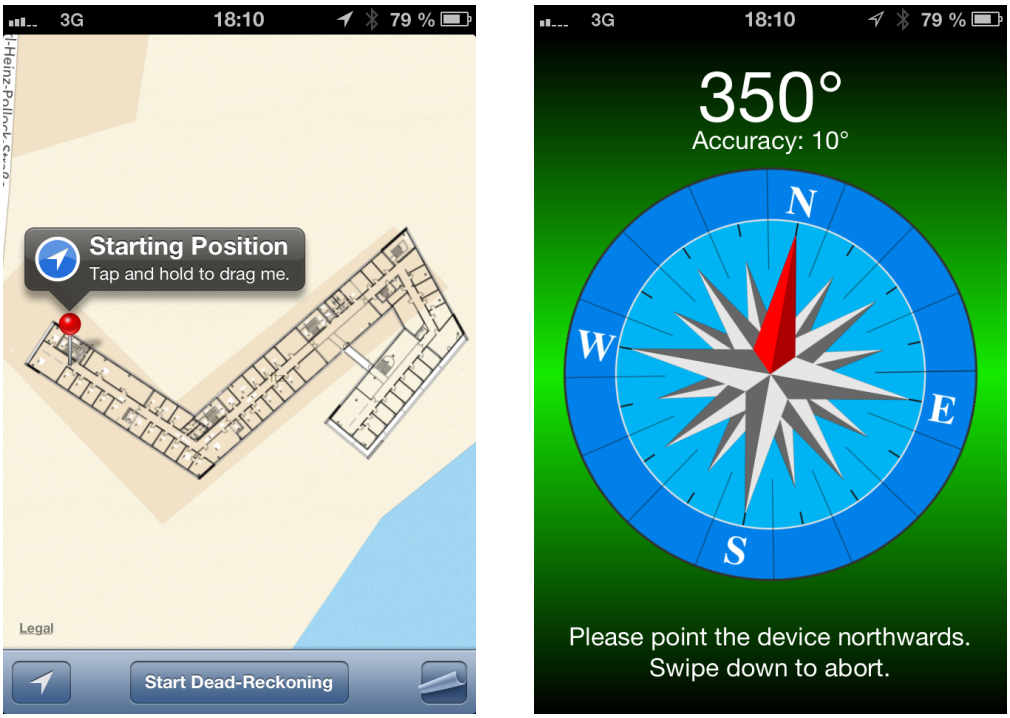


Figure 2.1.: System flow chart

After starting the app, the user is presented a world map (provided by the OS) with a pin showing the most recently acquired GPS position (Fig. 2.2a).

Since GPS works unreliably indoors, the pin is draggable by the user and he can correct his position aided by a transparent overlay of the floor plan.

Upon tapping the "Start Dead-Reckoning" button, the user is presented a calibration screen showing a live compass (Fig. 2.2b). The purpose of this screen is to acquire an initial "heading fix" in order to get an idea how the device is oriented. The acquired fix is then used to rotate the PDR coordinate system (see Sec. 2.1) by that amount. This amount of rotation can be corrected by the user at any time, which is often necessary due to the disturbances the magnetometer experiences indoors. Depending on the magnetometers reported accuracy, the user may be required to perform a calibration gesture with the phone (moving the phone along the shape of an ∞-sign).



(a) Initial screen, draggable starting pin **(b)** Heading fix acquiration

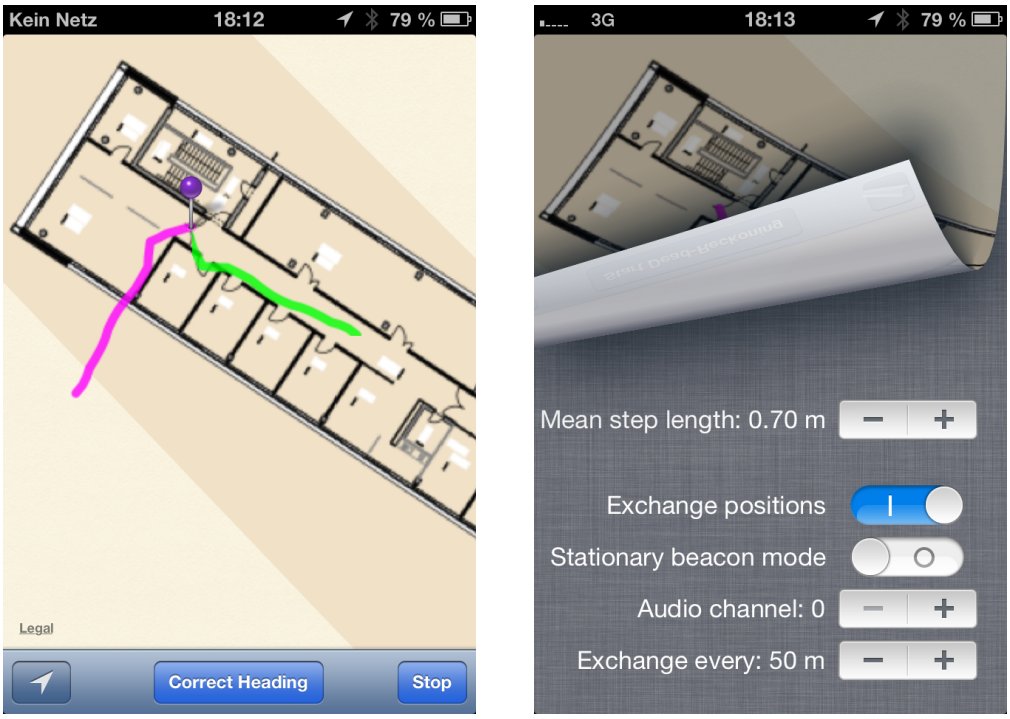
Figure 2.2.: Screenshots, before and after starting

After having successfully acquired the heading fix, the app starts the gyroscope and accelerometer sensors, which has the added advantage that these take the current orientation as a reference frame. Additionally, the camera is started in order to determine when the phone is placed into the trouser pocket.

In case the video the camera captures is "dark" enough, the motion sensors' data is passed on to the PDR algorithm, which tries to detect the steps of the

user. Furthermore, the Bluetooth module is activated and the app searches for other peers running *reckonMe*.

While walking, a "lock screen" is presented to the user, which apart from preventing the user from accidentally stopping the app, shows a log of recent events. After unlocking the screen, he can see the trace of the steps detected thus far on the map. Should he know his position better, he can again drag the pin marking his current position to where he presumably is and continue from there. In case he notices that the path appears rotated, he can manually rotate a part or the whole path by aligning it to corridors, for example (Fig. 2.3a). From then on, this amount of rotation is used to rotate the PDR coordinate system, so that the direction of future steps is more correct.



(a) Path (magenta), rotated live (green) **(b)** Settings screen

Figure 2.3.: Screenshots of heading correction mode, settings screen

If a peer is found, the app connects to that peer and negotiates the roles of each peer. These roles are the *sender* and *receiver* of inaudible sound patterns around 18 kHz, which are used to detect proximity. Once the receiver detects the sound patterns emitted by the sender, the devices are deemed proximate and an exchange of each others position estimate is performed via Bluetooth.

Prior to an exchange, there are thresholds to be checked, preventing exchanges from taking place too often. These include a certain amount of meters to be walked and to have at least one second passed in between exchanges. The exchanged estimates are averaged and weighed by their estimated error (growing linearly with recognized steps), such that both participants end up with a "new" and matching position (see Sec. 2.1.1). This collaborative step is then considered to be the current position and all following steps start from there.

On average, collaboration is beneficial, in that it lowers the total error for $N = 2$ (see Sec. 4.2.3) participants and can even exhibit an emergent behavior in case of many participants, yielding a constant error [5].

2.1. PDR Algorithm

The PDR algorithm was developed with the help of Dr. Kamil Kloch and has also been ported to Java (and hence, Android) by Matthias Schaff in [12]. It differs from many PDR algorithms in that it solely relies on device's orientation and acceleration, omitting the use of a magnetometer, which produces unreliable results in indoor scenarios.

A step's direction as well as its length can be determined by the device's orientation, whereas the step itself is delimited by peaks in the device's acceleration. Each device measures the changes in acceleration and rotation arounds its three axes, the device's reference frame, shown in figure 2.4.

On the iOS platform, there are several world reference frames to chose from, where `CMAttitudeReferenceFrameXArbitraryZVertical` has been chosen. The `CMAttitudeReferenceFrameXTrueNorthZVertical` reference frame would have been ideal for this purpose, as the x-axis is supposed to constantly point to "true" north, aided by sensor fusion with the device's magnetometer. Unfortunately, using this reference frame has the side-effect of sudden corrections to this reference frame. There is no API giving the app an opportunity to determine when and by which amount this reference frame is corrected, rendering it unusable for this approach.

Fortunately, all reference frames available have in common that their z-axis always points upwards, leaving the x- and y-axis in a horizontal plane. Since these point in arbitrary directions, the initial heading fix is acquired with the help of the user and all PDR steps are rotated by that amount at last. The iOS motion detection framework as well as the algorithm use quaternions, because

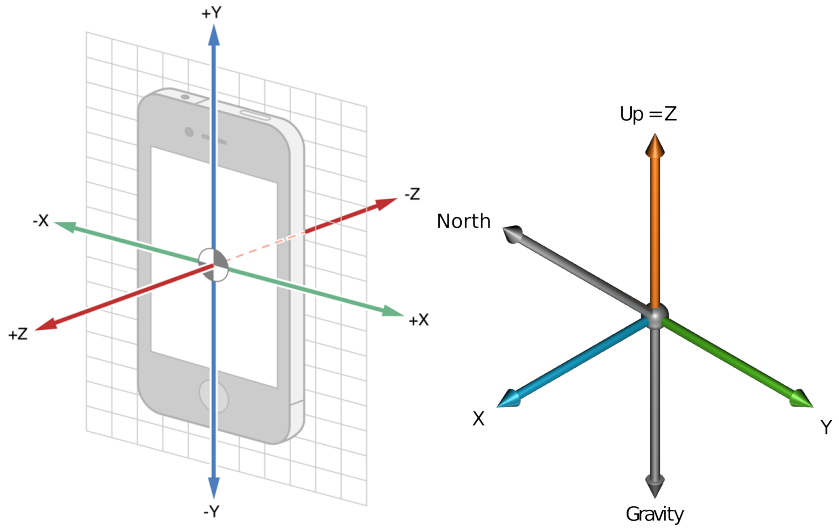


Figure 2.4.: Device reference frame, world reference frame `CMAttitudeReferenceFrameXTrueNorthZVertical`

these allow for a more elegant calculation of changes in device orientation and are not prone to "gimbal lock".

Preprocessing Initially, the raw quaternions, representing the device's attitude, need to be unwrapped. Wrapping occurs when a measurement exceeds the bounded interval of $[-1...1]$, which results in a sudden change of sign. Since we are interested in relative changes, these discontinuities have to be removed by unwrapping. Furthermore, the x-, y- and z-values of the acceleration vectors are combined by calculating their norm. Next, the quaternions as well as the normed acceleration are low-pass filtered using a Butterworth filter with an order of 10 with cut-off frequencies of $\frac{2}{25}$ and $\frac{1.6}{25}$, respectively. Afterwards, each smoothed quaternion q is used to rotate a gravity vector $g = (0, 0, -1)^T$, pointing to the center of the earth, into the device's reference frame, yielding $z = qgq^{-1}$.

Step Detection In order to detect steps, both, user acceleration and orientation are analyzed simultaneously. First, the z-axis gravity vectors z values are subject to peak detection. The detected peaks are then filtered using thresholds for the x- and y-axis of the gravity vectors and a threshold for the minimum required acceleration at that time. Doing this, changes in orientation that do not coincide with certain amount of acceleration are ruled out, because they can't be steps.

Each consecutive pair of peaks is furthermore checked for consisting of different types (positive followed by negative or vice versa) and for its temporal distance. If too much time passed in between peaks, it is discarded. Furthermore, the first peak's type is used determine the direction of the step.

Each of the remaining pair of peaks corresponds to a step, confined by the first and second peak of each pair. Its total rotation is calculated by multiplying the filtered quaternions corresponding to that peaks. The step length is calculated by taking the arc-cosine of the resulting quaternion's angle, multiplied by an average stride length. The step's direction is defined by the x- and y-component of the normalized quaternion, but needs to be rotated by -90°, as the vector part of the quaternion defines the rotation axis, which is perpendicular to the walking direction. However, this direction needs to be rotated once again by the heading fix acquired initially.

2.1.1. The Collaborative "Step"

After a successful exchange, each PDR algorithm receives the position estimate of the other peer. Let $p_a = (x_a, y_a, \delta_a)$ be the coordinates and corresponding deviation estimate of one peer and $p_b = (x_b, y_b, \delta_b)$ the estimates of the other peer in world coordinates. In the current implementation, δ initially starts with one and is simply increased by one for every detected step.

The new collaborative step $p_c = (x_c, y_c, \delta_c)$ from which both peers proceed computes as:

$$x_c = \frac{x_a\delta_b^2 + x_b\delta_a^2}{\delta_a^2 + \delta_b^2} \tag{2.1}$$

$$y_c = \frac{y_a\delta_b^2 + y_b\delta_a^2}{\delta_a^2 + \delta_b^2} \tag{2.2}$$

$$\delta_c = \frac{\delta_a\delta_b}{\sqrt{\delta_a^2 + \delta_b^2}} \tag{2.3}$$

It is worth noting that these equations strongly (to the square) favor smaller δ deviations to larger ones, in that a peer having walked for longer, strongly gravitates towards the other peer having walked less. This makes sense, as a peer that did not walk at all knows exactly where he is and hence should greatly contribute in improving the other peer's estimate. The stationary peer on the other hand, experiences only a small amount of "drag" from the peer with higher deviation. In the case of $\delta_a = \delta_b$, the new position simply becomes the arithmetic mean.

2.2. Audio-based Proximity Detection

First of all, it is worth noting that the audio-based proximity detection algorithm used is a slight variation of the algorithm published earlier by the author in [15]. Its evaluation in section 4.3 also refers to the algorithm used in [15], but the variation does not invalidate the results. In the following, it will become clear, why.

Experimenting with the speakers of an iPhone 4 and aiming for minimal annoyance of users, the author discovered that the frequency range of 18–20 kHz – being inaudible to most people – is applicable for sound emission as well as detection. The phone's speaker is able to produce sound in that range and the microphone's frequency response is fairly linear and big enough to be measured; these findings are consistent with [1].

Consider two devices D_A and D_B being in the pockets of two people walking relatively distant, but approaching each other. As soon as D_A "sees" D_B via Bluetooth, it connects to D_B, they negotiate a carrier frequency f and assign the roles of *sender* and *receiver*. Using these two roles is a concession to the iOS-platform, as it turned out that simultaneous playback and recording is artificially limited in playback volume by the OS and the delays introduced by dynamically switching between these modes were too high to be considered a viable option.

The sender starts emitting a sine wave oscillating at the negotiated frequency f. Sending does not take place continuously rather than being amplitude modulated. Amplitudes are determined by a certain pattern containing k binary entries, which represent intervals of length l_s, so the length of one pattern is $l_p = k \cdot l_s$. This approach is closely related to On-off keying (OOK), although in contrast to data transmission, the aim lies at detectability of the signal at all.

The receiver records the ambient sound and performs a Fast Fourier Transform (FFT) for every interval of length l_s of the input signal. Figure 2.5 shows the ideal pattern $[1, 1, 0, 1, 0, 0]$, which is being sent on a carrier frequency of 18 kHz, overlaid with the magnitudes of the corresponding FFT result for f of the signal received under ideal circumstances.

A sampling rate of 44.1 kHz and intervals of 512 samples have been chosen, which makes the interval length $l_s = \frac{44{,}100}{512} \approx 0,01161s$ and with $k = 6$ values per pattern, the pattern length is $l_p \approx 0,07$s. This pattern is rather short, as it allows for shorter window length and hence faster detection, while increasing the possibility to "slip through" in low-noise segments. Furthermore, variance in signal amplitude per window is lower using shorter windows in case the

persons move. The FFT has been chosen over a finite impulse response filter (FIR) because it allows to analyze several frequencies (i.e. another sender) at once at little additional cost. Additionally, this leverages the fact that iOS offers high-performance vector-based functions for performing FFTs.

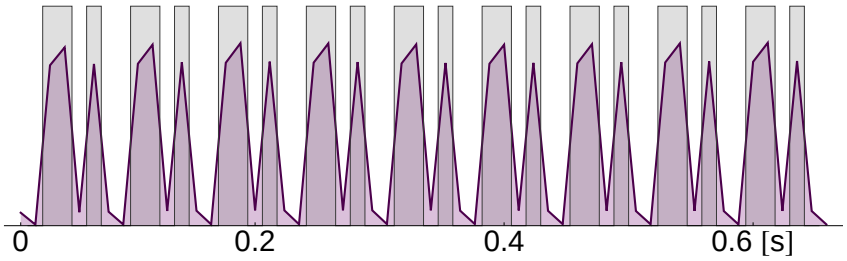


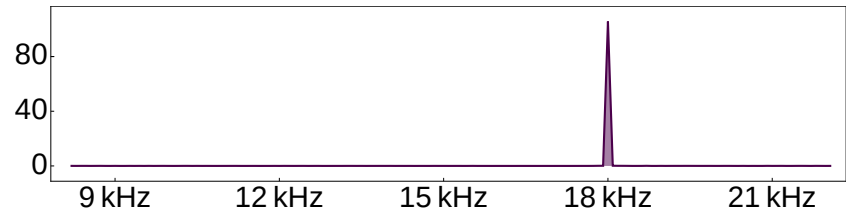
Figure 2.5.: An ideal sequence of patterns, overlaid with the recorded signal when speaker and microphone face each other in 10 cm distance.

As the FFT yields complex results and the interest lies in the signal's amplitude, their magnitude $m = |z_f|$ is computed for each FFT bin $z_f = (x, y) \in \mathbb{C}$, where f is a frequency of interest. The same is being done for an adjacent but unused frequency, which is taken as a sample of the overall noise level in that frequency range.

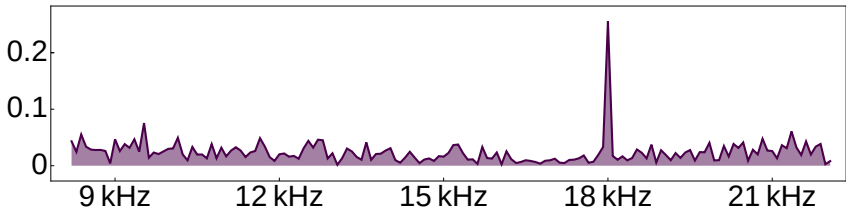
Noise, in this case, is mainly created by the phone rubbing against the pocket's fabric and is both, extremely loud and quite evenly distributed across adjacent frequencies. Figure 2.6 shows samples of frequency spectra as recorded by the receiver with both phones in different positions. As one can easily see, the signal in line of sight (see Fig. 2.6a) is already several orders of magnitude louder than the sample with both phones in the pocket and standing still (see Fig. 2.6b), which is still excellent compared to Fig. 2.6c, in which the signal is literally drowned in noise created by rubbing against the pocket. This kind of noise poses the most severe challenges to the detection approach.

Each k magnitudes of the carrier frequency, as well as the frequency used as a noise sample, are considered one window and are cross-correlated with the window earlier in time, yielding the correlation coefficients c_s for the signal and c_n for noise, respectively. The beauty of this approach lies in auto-correlating windows of length l_p, which are known to contain exactly one pattern, albeit offset by an unknown amount of time. Consequently, synching the receiver with the sender can be omitted and clock drift is mitigated, as it is quite small between two windows.

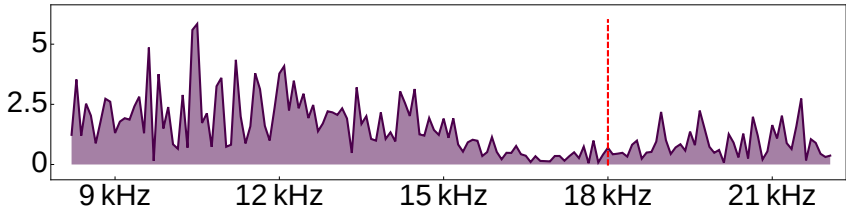
A new correlation measure $c_d = c_s - c_n$ is created, which approximates the amount by which the signal outgrows the noise. Subsequently, the signal auto-



(a) in line of sight 10 cm apart



(b) in pocket, standing 1 m apart



(c) in pocket, walking 1 m apart

Figure 2.6.: Samples of frequency spectra as recorded by the receiver, while both phones are positioned as stated in (a) – (c)

correlation c_s as well as c_d are averaged using sliding windows of different sizes to account for different characteristics:

- a_d is the sliding average of c_d over a larger window of $k_{a_d} = 16$ values in order to statistically collect traces of the signal in noise, while bursts of noise yielding small or negative values should even out over time
- a_s is the sliding average of c_s over a smaller window of $k_{a_s} = 8$ values, which – if high – represents a fairly low-noise situation in the short term

These two averages are subject to a simple thresholding in order to cover two scenarios, for which $t_H > t_L$:

1. A high-noise scenario, typically arising while walking. A detection candidate satisfies the following condition:

$$d_1 = (a_d > t_H) \vee (a_s > t_H), \tag{2.4}$$

which enforces candidates to have a relatively high autocorrelation despite noise in the long term *or* a relatively high autocorrelation in the short term, typically in between two steps.

2. A low-noise scenario, typically arising while not moving. A detection candidate satisfies the following condition:

$$d_2 = (a_d > t_L) \wedge (a_s > t_L), \tag{2.5}$$

which enforces candidates to have a moderate autocorrelation despite noise in the long-term *and* a moderate autocorrelation in short-term, which is the case when standing still or sitting, even when the signal is weak and mixed with moderate ambient noise.

The signal is considered partially detected if $d_p = d_1 \vee d_2$ is true. However, in the top-most abstraction layer, each 14 (because $l_p \cdot 14 \approx 1\text{s}$) values of d_p are counted and required to exceed yet another threshold, tailored for the two investigated situations: standing and walking. Constants are chosen carefully to almost entirely rule out false positives as well as successful detections too far away to be considered proximate.

2.2.1. Alteration for reckonMe

As mentioned earlier, an alteration of the above algorithm has been used in *reckonMe*. More precisely, the top-most abstraction layer of 14 detection candidates, which are subject to another thresholding, has been omitted. That is, one detection candidate is sufficient for an exchange to be initiated.

For [15], this layer had to be introduced in order to rule out false positives in a scenario where both persons stand farther apart. However, this standing scenario is not relevant to the PDR approach, where both participants are walking. Even worse, it posed a hurdle too high for the proximity detection of two walking participants.

3. Implementation

The implementation of *reckonMe* is an iPhone app, targeting iOS 5.1 and was developed using Xcode 4.6.3 on an iPhone 4. The experiments have been performed on two iPhone 4 running iOS 6.1, where it exhibited – even though it's CPU only has a single core – only 20 – 30 % processor usage while running. It is expected to work equally well on the succeeding iPhone models, but not on preceding ones, as earlier incarnations lack a gyroscope.

The app does not use any other third-party libraries than PROJ.4[1], which is an open-source geodesy library providing the functions used to convert between world coordinates (latitude, longitude in degrees) and their projected values (northing, easting in projected meters) in the Mercator projection used in Apple's mapping framework.

As is customary, the app is written in Objective-C – a superset of C offering a thin object-oriented layer – in an object-oriented manner using a Model-View-Controller pattern (MVC).

The following is a non-exhaustive list of implemented classes containing remarks on design choices and hurdles that had to be overcome during their implementation.

3.1. View and Controller Classes

FirstViewController manages the main view (as seen in Fig. 2.2a), which the user sees at first. Additionally, it manages the whole app's operation using a state machine. It is responsible for intercepting most user interactions and delegating the requested actions to the appropriate classes.

[1] https://trac.osgeo.org/proj/

MapViewProtocol is an interface definition (in Objective-C terms: protocol), which is implemented by both, `OutdoorMapView` and `IndoorMapView` and specifies the requirements both classes need to meet. Doing this allows for an easy exchange of the map of the whole world overlaid by a floor plan and an indoor floor plan constrained to a certain area. It requires its implementers to offer the functionality to

- draw the computed path on the map
- allow the user to specify and correct his position
- move the map's center to a specified location, in order to show a newly computed position estimate
- show a live rotatable copy of the path in order to let the user correct for a heading offset, around a freely choosable rotation center along the path (shown in Fig. 2.3a)

OutdoorMapView implements the functionality required in `MapViewProtocol` and manages an instance of `MKMapView`, which is the map view provided by the OS. Furthermore, it creates an instance of `FloorPlanOverlay`, which renders a given floor plan overlay on top of the map (see Fig. 2.2a). Due to its usage of the built-in map view, which is also used in Apple Maps, a new user is instantly familiar with the functionality of the map.

FloorPlanOverlayView provides the floor plan overlay, which can easily be used by providing either an PDF (using vector graphics) or bitmap image (in a format that offers transparency, such as PNG) containing the map, specifying its center in world coordinates and its scale in pixels per meter. Implementing this class has not been easy, as documentation on how the drawing on top of an `MKMapView` has to be implemented, is sparse.

Drawing is performed on a tile-basis, which has to take the scale of the floor plan, the current scale of the map, as well as the clipping of the drawing area to the appropriate tile into account.

In addition, custom drawing on iOS (in UIKit classes) requires the developer to flip the coordinate system expected by the Quartz drawing APIs (whose origin is lower-left, stemming from MacOS X), to the one used on iOS (whose origin is upper-left), further adding to the source of errors.

IndoorMapView offers the `MapViewProtocol` functionality and was implemented from scratch. In order to keep the memory footprint low, it offers tile-based drawing even of large custom maps. Implementing it, the author gained the fundamental drawing knowledge that greatly helped implementing `OutdoorMapView`.

CompassViewController simply shows a live rotating compass (Fig. 2.2b) and requests the user to hold his phone northwards for three seconds. This is not always easy given that the magnetometer frequently "jumps" in its heading estimation and exhibits an error of up to 90° in both directions (see Fig. 4.3). The purpose of this screen is to get an accurate heading fix to rotate the PDR coordinate system by.

AlertSoundPlayer is not a view, but also gives the user feedback about the current state of the app by letting the phone vibrate and by playing (pre-recorded) speech-synthesized messages. These include announcements like "starting", when the sensors are started in the pocket, or "exchanged positions" after a successful exchange. This functionality turned out to be especially useful, considering that the phone is carried in the pocket most of the time.

3.2. PDR Classes

PantsPocketDetector uses the built-in camera to determine whether it is "dark" around the phone, assuming it is in the pocket by then. It captures a picture every second with the front-facing camera and computes its average luminance as well as the luminance's standard deviation. In case the average luminance falls below 0.1 and the standard deviation below 0.023 (meaning there is next to nothing to see, except for noise), the pocket status is deemed detected.

This class' purpose is to defer the detection of steps by `PDRController`, through cutting it off from motion sensor values, until they are actually to be analyzed for steps. Earlier incarnations of this class also featured the device's infrared-sensor (proximity sensor) next to the earphone, but it turned out (which is undocumented) that this silences the device's speaker, rendering this method unsuitable for the audio-based proximity detection. Presumably, this is done by iOS in order to spare the user the pain of a loud speaker, when holding the phone to the ear.

PDRController is a real-time adapted implementation of the PDR algorithm described in section 2.1, which is run every second. The class is written in Objective-C++, a dialect allowing to write parts of the code in C++, which has been used for its broader range of and more efficient collection classes available in its Standard Library.

It is worth noting that using Objective-C++ mandates that all calling classes need to be written (at least formally) in Objective-C++, too. This can be achieved by changing their file extension from the Objective-C standard ".m" to ".mm", which can be quite time-consuming and leads to obscure compiler errors if a class is overlooked. Once these compile time hurdles are taken on the other hand, there are no runtime drawbacks in mingling Objective-C with Objective-C++.

In order to keep the code simple, `PDRController` makes use of the optimized methods of OpenGL for its quaternion-based calculations. Furthermore, it uses overlapping sliding windows for filtering its data in order to minimize boundary effects caused by filtering. The class maintains two PDR traces: one *raw* trace, consisting of the mere unrotated steps, which is only recorded and a *collaborative* trace containing all changes made, including collaborative or manual position corrections and heading corrections, which is recorded as well as shown on the map.

LocationEntry and its subclass `AbsoluteLocationEntry` encapsulate a PDR step in either local or world coordinates, respectively, using their members:

timestamp a UNIX timestamp in milliseconds since beginning of *The Epoch*

eastingDelta the value on the x-axis in meters

northingDelta the value on the y-axis in meters

deviation the deviation δ as defined in section 2.1

and, additionally in `AbsoluteLocationEntry`:

origin a (latitude, longitude) tuple that defines the origin of the coordinate system in world coordinates, which is also the starting point of the PDR algorithm

While this approach seems pretty straightforward, there is one catch in `AbsoluteLocationEntry`: the fact that the Mercator projection used in the mapping frameworks, although conserving straight segments and angles,

greatly distorts size (up to infinity on the poles) the further away from the Equator.

However, this class transparently takes this into account. The scale distortion is given implicitly by the latitude and the scale factor calculates as:

$$m_f = \frac{1}{\cos(\text{latitude})} \tag{3.1}$$

which, for Passau (latitude $\approx 48.6°$), results in $m_f \approx 1.51$.

Let $m(\text{latitude, longitude}) \rightarrow (\text{northing, easting})$ be the Mercator projection and $p = (o_{\text{lat}}, o_{\text{long}}, d_e, d_n)$ an AbsoluteLocationEntry with o as origin and easting and northing deltas d. Its absolute location $(a_{\text{lat}}, a_{\text{long}})$ in world coordinates computes as:

$$(o_n, o_e) = m(o_{\text{lat}}, o_{\text{long}}) \tag{3.2}$$
$$a_n = o_n + m_f d_n \tag{3.3}$$
$$a_e = o_e + m_f d_e \tag{3.4}$$
$$(a_{\text{lat}}, a_{\text{long}}) = m^{-1}(a_n, a_e) \tag{3.5}$$

The coordinate system in which the PDR steps are computed hence needs to be scaled up by m_f before being shown on the world map. While this also induces a small error when exchanging instances of `AbsoluteLocationEntry`, this error is negligible compared to the much bigger inherent error introduced by dead-reckoning.

FileWriter is a class that records the whole PDR session into plain text files, prefixed with date and time, into a directory named by date and time. This includes the following data and its timestamps:

- manual position corrections
- manual heading corrections
- position exchanges
- the *raw* PDR trace
- several *collaborative* PDR traces. Each file is written to until a change is made. From then on, a new file, containing the newly modified, whole trace is started. Changes include a manual position or heading correction and position exchanges.

as well as the following sensor data:

- accelerometer data
- gyroscope data
- magnetometer data
- GPS positions (which can be seen as a ground truth when used outdoors)

Furthermore, at the end of a session, a screenshot is taken programmatically in order to give the person analyzing the recordings a glimpse of the result.

The files are placed into the app's document directory, where they can easily be obtained using iTunes File Sharing by connecting the phone via a USB cable or by enabling WiFi sync. Within the app, there is no possibility to delete completed sessions, as a precaution for accidental deletion.

3.3. Peer-to-Peer Classes

P2PestimateExchange is responsible for exchanging position estimates via Bluetooth and controls an instance of `SoundDetector` for audio-based proximity detection.

This class makes use of a `GKSession`, which is part of the GameKit API, for initiating and managing a connection. As the name suggests, it is mainly aimed to be used for ad-hoc sessions connecting two or more gamers. Unfortunately, this was – by the time of implementation – the only API available to directly connect to other devices, as direct control of networking hardware was impossible on iOS. This has changed with the introduction of the CoreBluetooth framework, though it requires Bluetooth 4.0 Low Energy hardware, which the author had no access to.

While the GameKit API is relatively easy to use, it is rather poorly documented and limited in its capabilities, which imposes several drawbacks. Firstly, there is no choice between using Bluetooth and WiFi. GameKit appears to give preference to WiFi, so in order to force Bluetooth usage, the user has to deactivate WiFi and activate Bluetooth in the settings app. While it may make sense to use WiFi in smaller environments, it does not once the devices join different networks, effectively isolating themselves from each other. Furthermore, there is no means of finding out the signal strength (RSSI) of a visible device, making it only useful for coarse proximity detection.

Then again, this sparked the author's imagination and led to the idea of using sound for a finer-grained proximity detection, as working GameKit connections using Bluetooth with a distance (though unobstructed) of up to 80 m indoors have been experienced.

With GameKit being relatively high-level, it became obvious that what is called an *available* status, already requires a physical connection, which is automatically established beforehand by the framework. This has the added benefit of the peer's freely assignable name being known before actually *connecting* in GameKit terminology. On the other hand, this can lead to a fragmentation into groups of peers that are only visible to each other within the same group. It may well be the case that GameKit establishes a Bluetooth personal area network (PAN or piconet) underneath, which is limited to up to eight devices, one of which acts as a master whereas the others act as slaves.

Another observation (since undocumented) made is that a successfully established connection, once disconnected, excludes the peers from "seeing" each other again and hence impairs their possibility to reconnect again. In order to solve this, the destruction of the running `GKSession` is required and a new instance has to be created. What is more, a peer might get stuck in connecting without triggering a timeout and calling the designated callbacks, which again, requires the re-creation of a session. While this certainly leaves an uneasy feeling, these workarounds worked quite well in practice.

The implemented process for an exchange is the following: Once a peer "sees" another peer and the peer is allowed to exchange (again) by having walked a sufficient number of steps, it starts either detection or emission in `SoundDetector` on the channel the sender advertises, which is implicitly encoded in each peers' name.

At the moment, the channel number is fixed and can be changed in the settings (see Fig. 2.3b). Moreover, the role assignment is also fixed, in that peers on even channels act as the receiver, whereas peers using odd channels acts as senders. More advanced dynamic role and channel assignments can easily be added, but were omitted since they were not needed in the simple experiments performed for the evaluation and would have only introduced a new source of errors.

Once the receiver detects the sender, it initiates an exchange by connecting and sending its own estimate. The sender, on the other hand, acknowledges the reception of the estimate and sends its own estimate. Upon reception, the receiver acknowledges the reception of the acknowledgement and vice versa. Now, both peers can be sure that the other peer received their estimate *and*

they can be sure they mutually know that the other peer received their acknowledgement. The exchanged positions are passed to `PDRController` and the peers disconnect.

As can be seen in the screenshot (Fig. 2.3b), a *beacon mode* has also been implemented, but was omitted in the evaluation, as it represents a kind of infrastructure which was sought to avoid. A phone running *reckonMe* in beacon mode would act as the receiver always exchange its fixed position without altering its own, thereby essentially forcing its position onto the detected peer. As the detection without the noise introduced by walking works very well, this is a low-hanging fruit.

SoundDetector is a straight-forward implementation of the audio-based proximity detection algorithm described in section 2.2. Needing direct access to the buffers played and recorded, it makes use of the low-level C-API `AudioSession` of the CoreAudio framework. Working with this API was hard at first, as its parts are well documented, but not their interplay.

Furthermore, the roles of sender and receiver are a concession made to the iOS platform. Upon creating an AudioSession, a category describing the intended usage needs to be specified, of which `kAudioSessionCategory_PlayAndRecord` would have been the category of choice, as playing *and* recording simultaneously are intended. However, it turned out that (as undocumented) this category automatically drastically lowers the playback volume, presumably doing this in order to prevent recording too much of its own audio. As a result, different roles using a `kAudioSessionCategory_PlayAndRecord` for the receiver and a `kAudioSessionCategory_MediaPlayback` for the sender have been chosen, allowing the playback of the sound pattern at maximum volume. Dynamically switching between those categories has been tested, but tearing down and re-creating the complete audio session took too long (several seconds) to be a viable option.

In order avoid "cracking" noises when playing back the on/off pattern, each change is linearly faded in and out for 16 of each 512 samples, which makes the cracking only audible when holding the speaker close to the ear.

Analyzing the audio data on the receiver's side makes extensive use of vector-based functions of the Accelerate framework (prefixed with *vDSP*), this includes the Fast Fourier Transform (FFT) as well as several statistics functions being used. Using a FFT instead of a Finite Impulse Response (FIR) filter has the benefit of being able to analyze all frequencies, or more precisely, all frequency intervals ("bins") at once. There are seven designated channels of which the last one is taken as the noise sample:

channel	**FFT bin**	**Hz**
0	105	18087
1	108	18604
2	111	19121
3	114	19638
4	117	20155
5	120	20671
6	123	21188

With the channel's frequencies lying three FFT bins apart, they are clearly distinguishable while all being inaudible and within the device's capabilities in terms of sampling rate as well as frequency response. Although other channels as 0 have not been tested as thoroughly, small tests indicated a very similar frequency response in terms of playback through the speaker and recording using the microphone.

4. Evaluation

This chapter evaluates the performance of the PDR algorithm and the audio-based proximity sensing algorithm in several experiments. At first, the PDR results of a single person are evaluated (Sec. 4.1), in order to show the accomplishments as well as the pitfalls of the approach.

Next, the evaluation of the PDR results of two persons – considered as individuals – are evaluated (Sec. 4.2.1), followed by the evaluation of their collaboration (Sec. 4.2.2).

Furthermore, a comparison of individual and collaborative performance is performed (Sec. 4.2.3), which clearly shows the gains of collaboration. Lastly, the performance of the audio-based proximity algorithm is evaluated in experiments independent of its PDR deployment, in section 4.3.

4.1. PDR of One Person

In order to assess the performance of the PDR algorithm, three different routes (see Figs. 4.1a, 4.1b, 4.2) of varying lengths within an office building (ITZ of University of Passau) have been conceived:

Route	**length in m**
R_1	55
R_2	84
R_3	197

With the help of the floor plan and its known scale, the length of each route could be pre-determined and was verified using a laser rangefinder. Each route was walked 30 times by the author, amounting to a total distance of over 10 km.

Although the app works with world coordinates, the starting and ending point of each route were chosen to be the same. This facilitates the evaluation of the

(a) R_1, length: 55 m **(b)** R_2, length: 84 m

Figure 4.1.: R_1 and R_2, heading correction at the red cross

Figure 4.2.: R_3, length: 197 m, heading correction at the red cross

displacement, as it coincides with the distance from starting to ending point. As a result, usage errors, introduced by not accurately setting the starting point and comparing the ending to a fixed point, are ruled out.

As with many buildings, the ITZ's steel and glass architecture imposes significant disturbances on the device's magnetometer; hence, its usage is constrained to the single heading fix at the beginning of a run, requiring human interaction and often a calibration gesture.

Even with the aforementioned recalibration, the magnetometer produces very unreliable results. Therefore, a manual heading correction, that is, a rotation of the path computed so far and the currently used coordinate system, is performed, usually at the end of the first straight line. Figure 4.3 demonstrates the necessity for this manual intervention, it shows the same 30 traces for R_1 with and without the heading correction. The correction process is aided by the display of the floor plan on screen and an in-place, live rotation of the path, which makes it easy to align the path along the aisles.

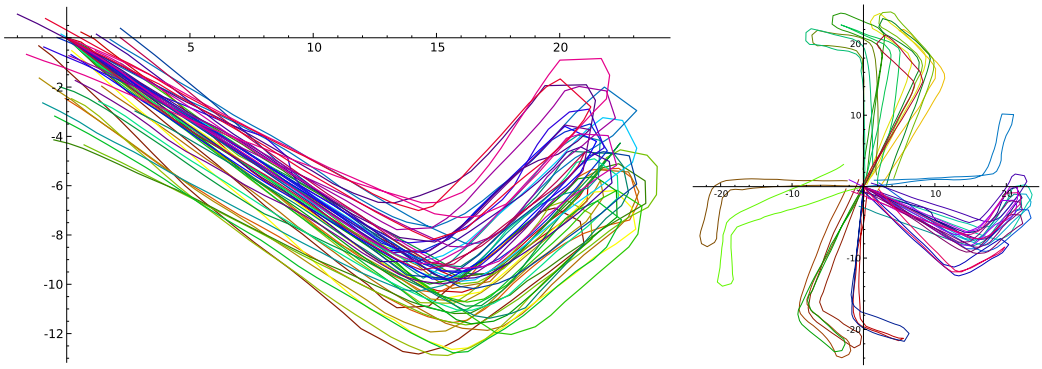
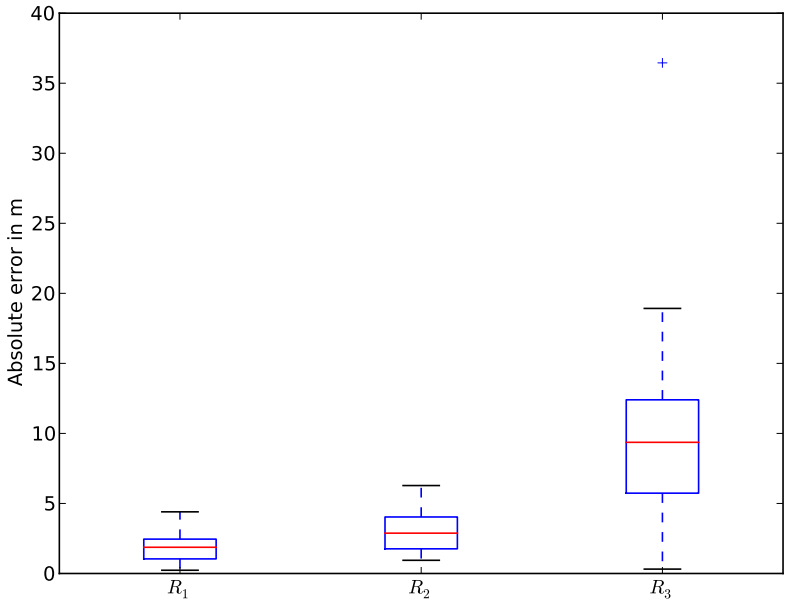


Figure 4.3.: All traces for R_1, with and without heading correction

The results of these runs are quite promising. As shown in figure 4.4, on average, the absolute displacement per run lies around 5 % of the path length.

While the total displacement appears to be growing linearly with length of the path, the path length estimation seems to be an exception at first glance, regarding figure 4.5. On a closer look, however, it becomes consistent with reality. Closed doors had to be passed while walking R_2 and R_3, which induces a constant number of additional steps taken and recognized.

In the case of R_2, there are four closed doors to open, two of which have to be pulled. While having to push a door does not necessarily slow the person down or forces him to make a step sidewards, pulling does, as seen in figure 4.6. The trace is to be followed clockwise from the origin. It clearly shows a "dent" where the doors had to be pulled open and the shorter strides made beforehand (less distance between consecutive points).



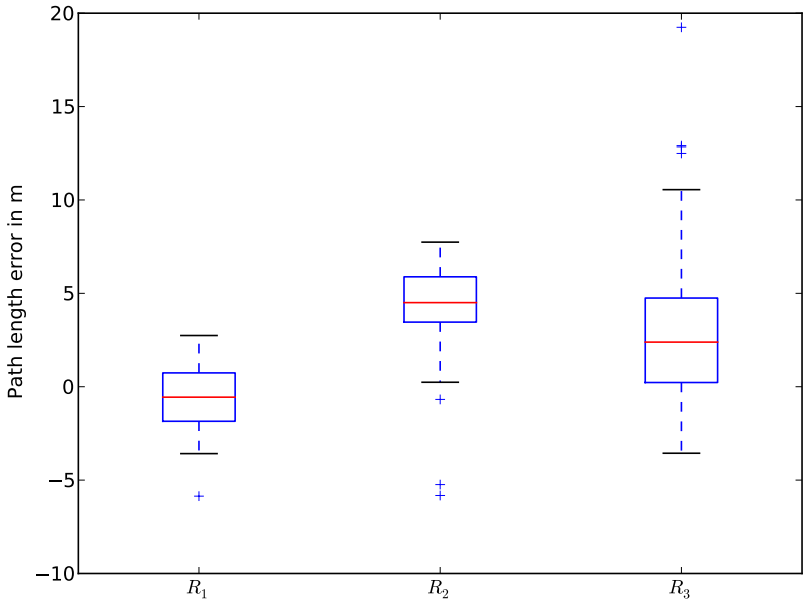
	l	$\bar{x}$	s	$x_{\min}$	Q_1	Q_2	Q_3	$x_{\max}$
R_1	55	1.89	1.14	0.23	1.04	1.87	2.45	4.40
R_2	84	3.06	1.52	0.94	1.76	2.88	4.03	6.28
R_3	197	10.00	6.93	0.31	5.73	9.36	12.40	36.45

Figure 4.4.: Statistics of absolute displacement error per run in m

R_3 on the other hand, has been walked on the ground floor[1], which involved one door passed twice. While this alone introduces an over-estimation of two meters, the inaccuracy increases with length due to not exactly walking the shortest path, i. e. by avoiding obstacles or other people.

As figure 4.7 indicates, the major cause of error for the PDR system is angular error. The device's gyroscope exhibits a small but persistent drift, which can neither be detected programmatically nor corrected. In order to assess the PDR system's angular error, each trace's beginning and ending segments (i. e. the first and last fifth of the trace) have been fitted using linear regression analysis.

[1] in contrast to the floor plan, which shows the first floor, as a floor plan of the ground floor was not available



	l	$\bar{x}$	s	$x_{\min}$	Q_1	Q_2	Q_3	$x_{\max}$
R_1	55	-0.63	1.89	-5.86	-1.85	-0.56	0.74	2.74
R_2	84	3.94	3.21	-5.82	3.46	4.50	5.88	7.74
R_3	197	3.84	5.26	-3.56	0.23	2.39	4.75	19.24

Figure 4.5.: Statistics of path length error in m

Let m_b be the slope of the path's beginning segment and m_e the slope of its ending segment, the (smaller) angle θ between these two lines satisfies the following equations:

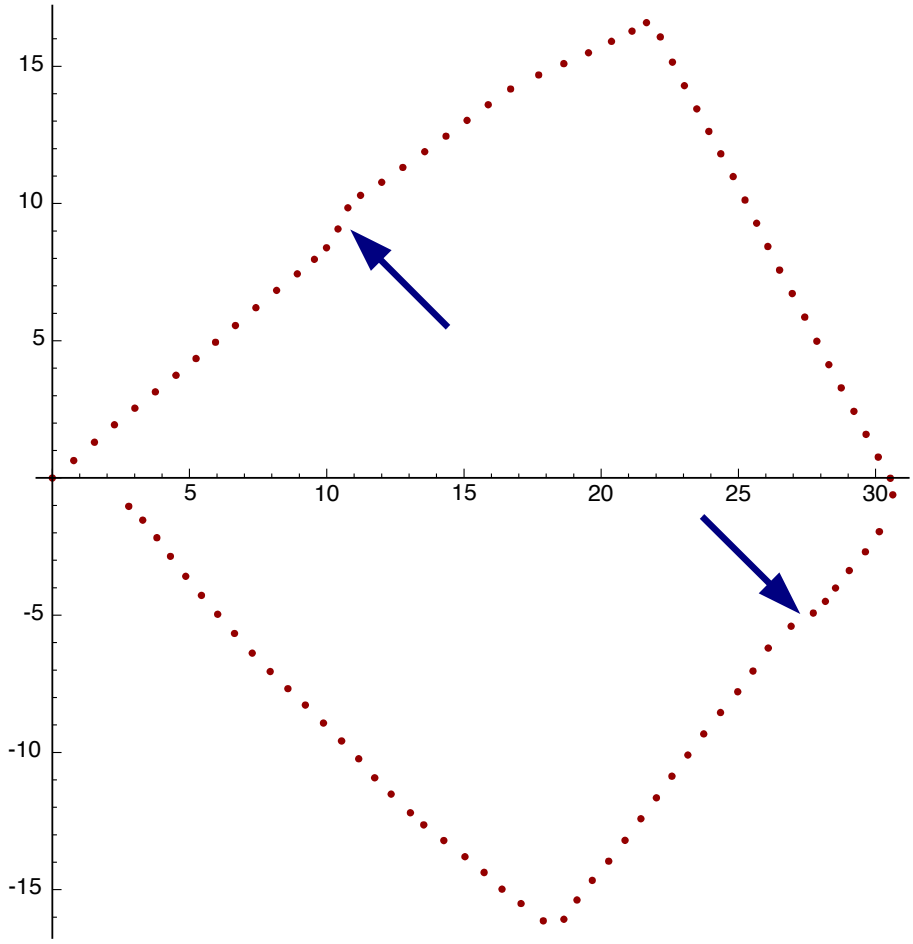


Figure 4.6.: A trace of R_2, arrows show doors that had to be pulled open

$$\tan\theta = \left|\frac{m_b - m_e}{1 + m_b m_e}\right| \tag{4.1}$$

$$\Rightarrow \quad \theta = \arctan\left|\frac{m_b - m_e}{1 + m_b m_e}\right| \tag{4.2}$$

Due to using the absolute value of the fraction, θ its always positive and thusly lacks "direction". To make up for this shortcoming, θ_d is defined as:

$$\theta_d = \begin{cases} \theta & \text{if } m_b < m_e \\ 1.413 - \theta & \text{if considering } R_2 \\ -\theta & \text{otherwise} \end{cases} \tag{4.3}$$

Ideally, θ should be zero for R_1 and R_3 and $81° \approx 1.413\,\text{rad}$ for R_2, hence the subtraction. Figure 4.8 shows two traces of R_2, the fitted lines and each θ_d.

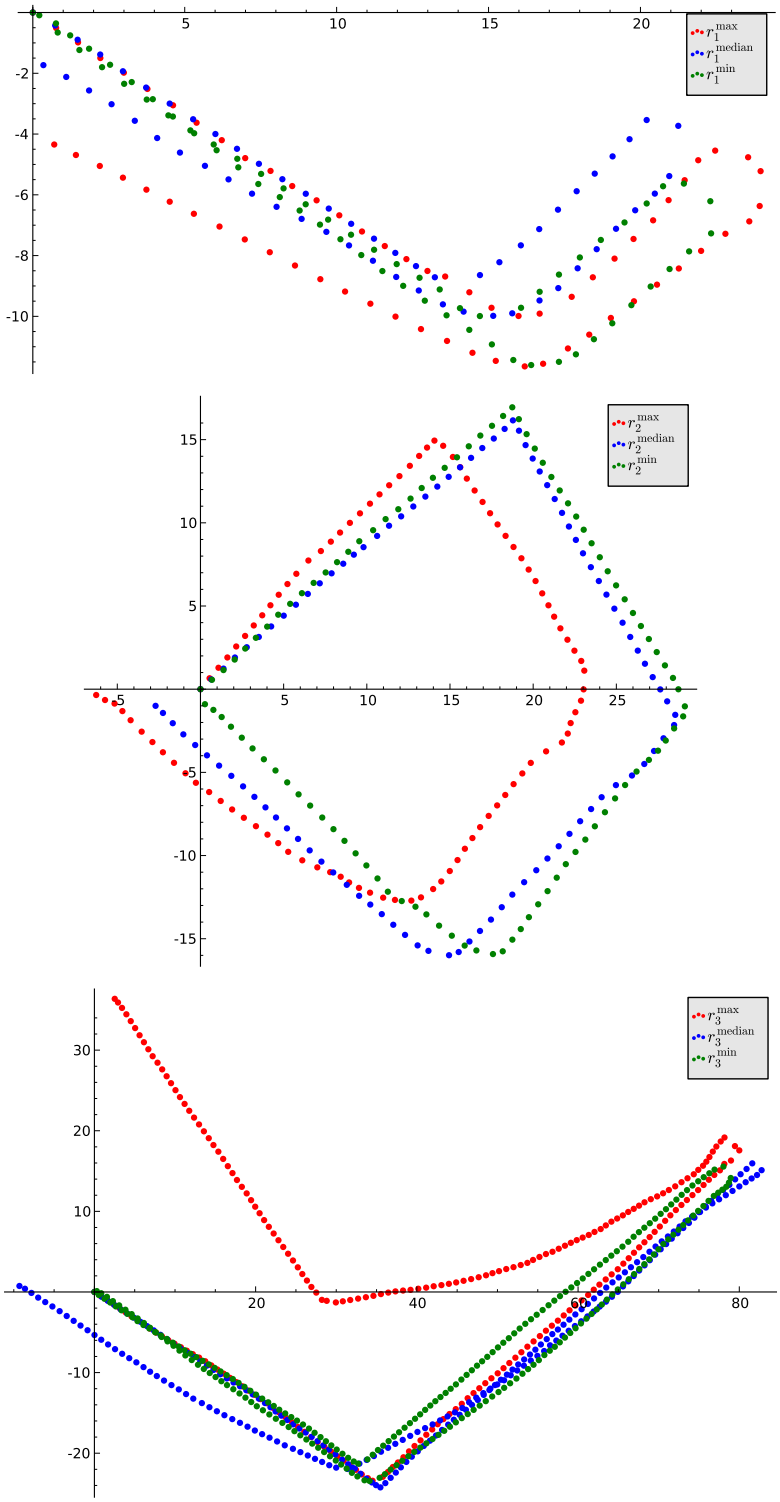


Figure 4.7.: The range of absolute error for each route exemplified

Negative values of θ_d indicate an ending of the trace that is rotated clockwise with respect to the beginning. Counterintuitively, a negative θ_d for R_2 indicates an *overestimation* of the angle by that amount and positive values an *underestimation*, respectively.

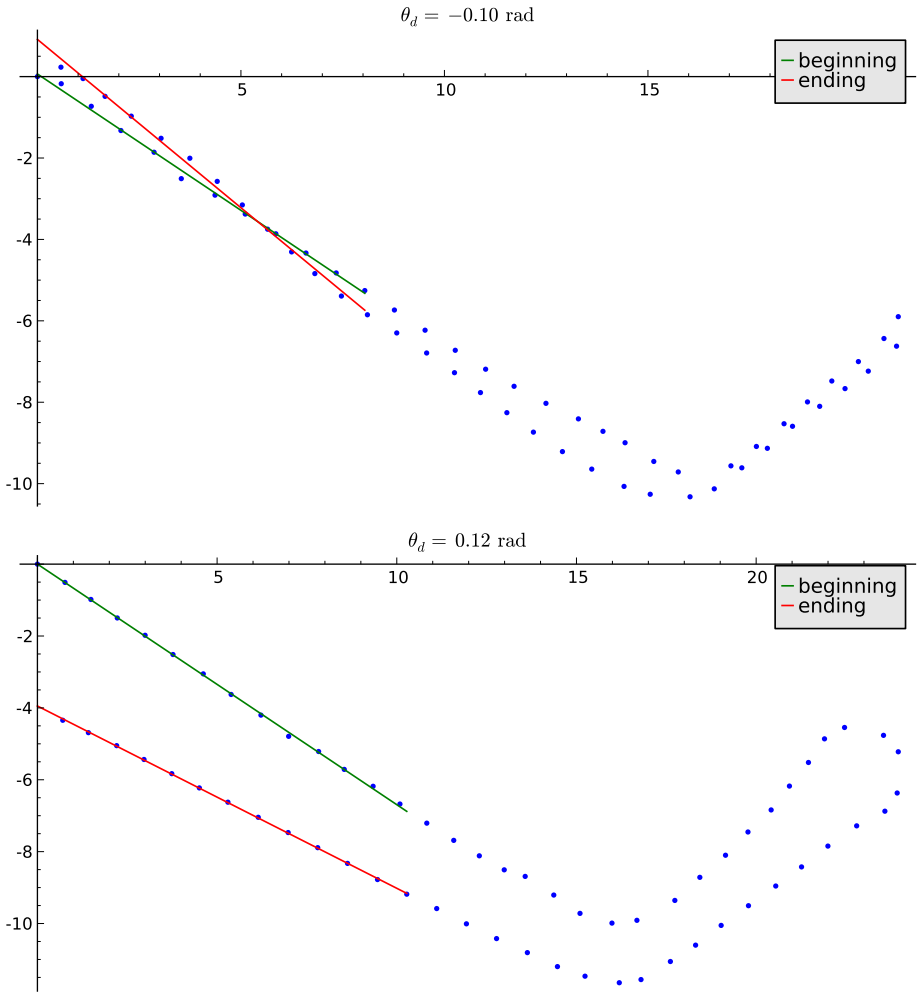
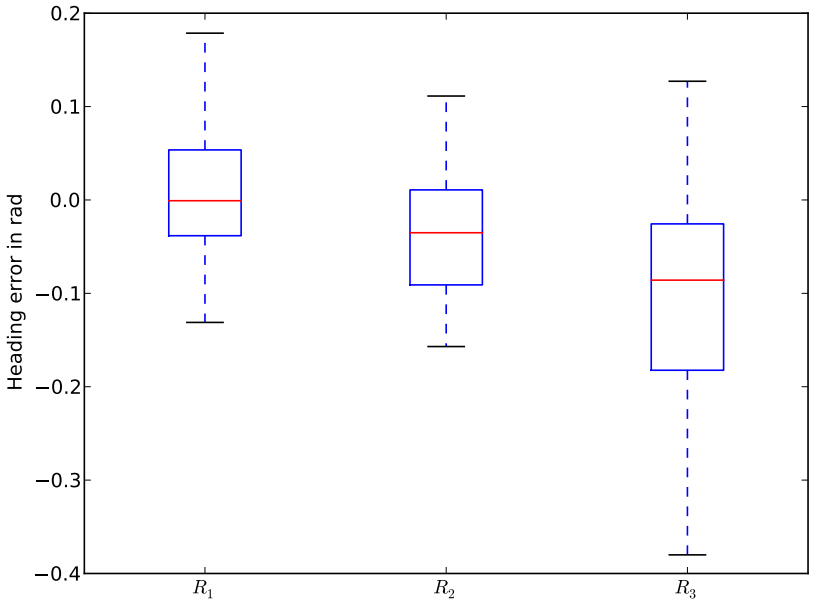


Figure 4.8.: Fitted lines and angles between beginnings and endings of R_1

It is worth noting that a larger angle deviation does not necessarily induce a larger displacement of the ending, as the beginning and ending lines may be "cancelling" each other out, as can be seen in the upper trace of figure 4.8. In fact, the angle deviation is a direct result of accumulated gyroscope drift in either direction. Furthermore, this drift can be weakened or strengthened by the direction in which the participant turns at the end of R_1 and R_3 or which trouser pocket the phone is in, factors which have not been controlled for.



	l	$\bar{x}$	s	$x_{\min}$	Q_1	Q_2	Q_3	$x_{\max}$
R_1	55	0.01	0.07	-0.13	-0.04	-0.00	0.05	0.18
R_2	84	-0.04	0.07	-0.16	-0.09	-0.04	0.01	0.11
R_3	197	-0.11	0.13	-0.38	-0.18	-0.09	-0.03	0.13

Figure 4.9.: Statistics of angular error in rad

As shown in figure 4.9, angular error grows with the path's length due to its accumulating nature. Interestingly, for routes R_1 and R_2 the distribution is relatively centered around zero, whereas for R_3 it is skewed towards negative values. A possible explanation is an underestimation of the last two turns, which have mostly been performed by turning right, whereas the first turn is a turn left. This imbalance or a general drift to the left (due to wearing the phone in a particular pocket) are the most likely explanations.

4.2. Collaborative PDR of Two Persons

To assess the collaborative PDR performance, again, three routes in the same building (ITZ) have been conceived and walked simultaneously 30 times each by the author and another person. These routes have the following lengths:

Route	length in m
C_1	97
C_2	84
C_3	197

which amounts to total of over 22 km (11 km per person) walked. It is worth noting that C_2 and C_3 are equivalent to their single counterparts R_2 and R_3, whereas route C_1 is novel.

(a) C_1, length: 97 m, exchange after 73 m on second meeting (pink ellipse)

(b) C_2, length: 84 m, possible exchanges at the middle and the end (pink circles)

Figure 4.10.: C_1 and C_2, heading corrections at the at the red crosses

On route C_1 (Fig. 4.10a) the participants start from opposite ends of a corridor, coordinated by a hand gesture. After walking half of the corridor, they stop to correct their headings individually. In order to prevent a potential exchange at that point, the threshold of required meters to be walked is set to 50 m. The participants part ways and walk to the end of the corridor, turn around and meet each other again in the middle of the corridor, where an

exchange takes place. Subsequently, they walk to their initial position and stop the recording to start anew.

On route C_2 (Fig. 4.10b), the participants start from the same initial position at the same time, but walk the route in different directions. Both correct their heading at the ends of their first straight segment and continue walking. Around half of the route, they meet for the first time and possibly experience an exchange. They continue walking only to meet again at the end and – waiting for each other if required – have the chance for a second exchange, before stopping and starting anew.

Figure 4.11.: C_3, length: 197 m, heading corrections after 40 m (red cross), multiple potential exchanges while walking together (at most every 60 m)

For route C_3 (Fig. 4.11) however, the objective is different: The two participants walk next to each other and exchanges should occur "under way". Typically, this results in exchanges more frequently taking place near doors or at the end of the route when the participants turn around. The threshold of required meters to be walked is set to 60 m, allowing a maximum of three exchanges taking place in the best case.

While conducting the experiments, it has been made sure that each run had at least one exchange of PDR positions. The audio-based proximity sensing has been aided – in case the exchange did not take place already while approaching – by standing still for about one second on routes C_1 and C_2. This practice resulted in a certain detection followed by an exchange (Sec. 4.3). In the rare

case of an error (mostly due to Bluetooth connectivity issues), the particular run was discarded and repeated.

For the sake of simplicity and better comparability of PDR results, the roles of sender and receiver have been fixed beforehand. The author has always been the receiver, denoted in the following by C_x^R, $x \in \{1, 2, 3\}$, whereas the other participant acted as the sender (C_x^S). Doing this does not implicate a loss of generality, as it is irrelevant which device detected the other because the following exchange process remains the same and only runs with a successful exchange are evaluated.

4.2.1. Individual PDR Performance

Firstly, the PDR results of each participant are analyzed individually. Since the roles were fixed, the results of each participant can be examined separately. Doing this makes them comparable to the results obtained in section 4.1.

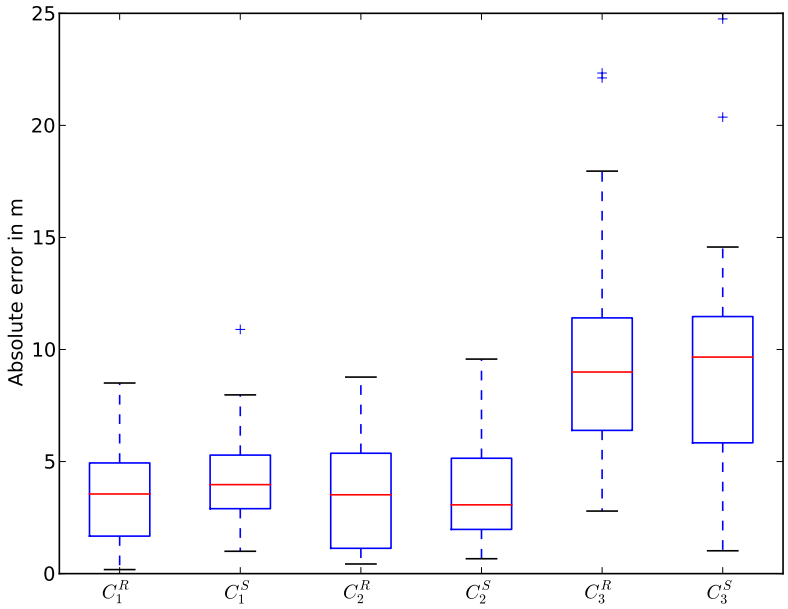
As expected, the absolute displacement errors (Fig. 4.12) are quite similar to the results achieved when walking alone (Fig. 4.4), regarding $C_2 \approx R_2$ and $C_3 \approx R_3$. On average, the total displacement error amounts to 4 - 5 % of the total distance walked. It is worth noting that both participants R (the author) and S appear to have achieved quite similar results. However, this is not the case, as shown in the following.

Figure 4.13 makes obvious, why. The stride lengths, and hence, the path lengths of S are drastically underestimated. On average, S falls short of up to a fourth of the path's length. As the designed paths end where they start, the measure of absolute displacement is inherently rotation- and scale-invariant. To an untrained eye, like S was, the shorter paths did not become obvious while conducting the experiments.

A quick look at figure 4.14 rules out the hypothesis of not enough steps being detected. The algorithm detected even more steps for S than for R.

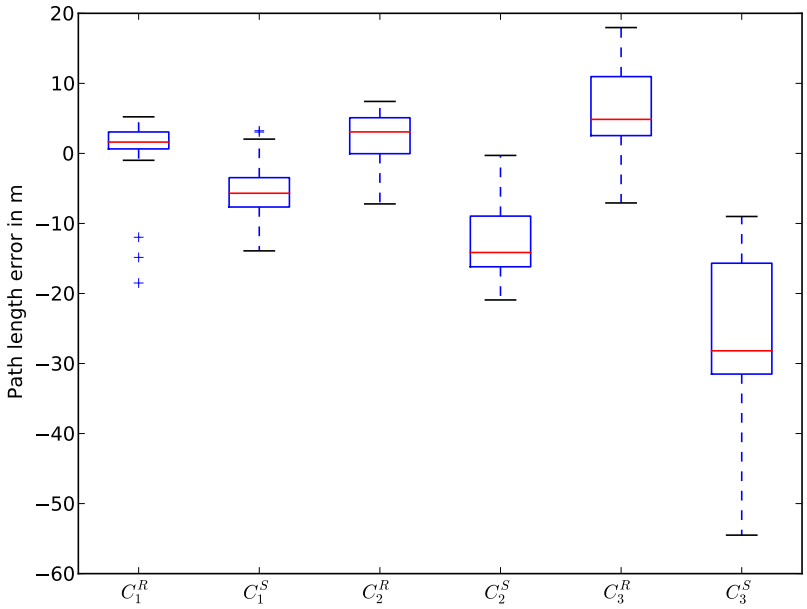
Hence, the stride length must have been underestimated. Table 4.1 reveals that, over all routes, at least three out of four strides have been detected to have a length of 0.62 m.

In hindsight, the most likely explanation becomes clear: S did not wear a trouser with a sufficiently "tight" pocket. As a result, the acceleration peaks, which are used to detect the beginning and ending of a step, were not as pronounced as expected or disturbed by additional movement of the phone in the pocket. The detected steps exhibited too narrow an angle, so that the



	l	$\bar{x}$	s	$x_{\min}$	Q_1	Q_2	Q_3	$x_{\max}$
C_1^R	97	3.48	2.24	0.18	1.67	3.55	4.94	8.50
C_1^S	97	4.34	2.10	1.00	2.90	3.97	5.29	10.90
C_2^R	84	3.50	2.42	0.43	1.13	3.52	5.37	8.77
C_2^S	84	3.67	2.35	0.66	1.97	3.07	5.15	9.57
C_3^R	197	9.94	4.75	2.79	6.39	8.99	11.41	22.34
C_3^S	197	9.13	5.33	1.02	5.84	9.66	11.47	24.75

Figure 4.12.: Statistics of absolute displacement error per run in m



	l	$\bar{x}$	s	$x_{\min}$	Q_1	Q_2	Q_3	$x_{\max}$
C_1^R	97	0.46	5.58	-18.51	0.64	1.62	3.06	5.22
C_1^S	97	-5.39	4.22	-13.92	-7.66	-5.69	-3.46	3.23
C_2^R	84	2.27	3.77	-7.21	-0.04	3.07	5.09	7.42
C_2^S	84	-12.68	4.98	-20.92	-16.19	-14.15	-8.95	-0.29
C_3^R	197	6.09	6.08	-7.08	2.54	4.86	10.96	17.97
C_3^S	197	-25.87	10.14	-54.50	-31.51	-28.18	-15.68	-9.01

Figure 4.13.: Statistics of path length error in m

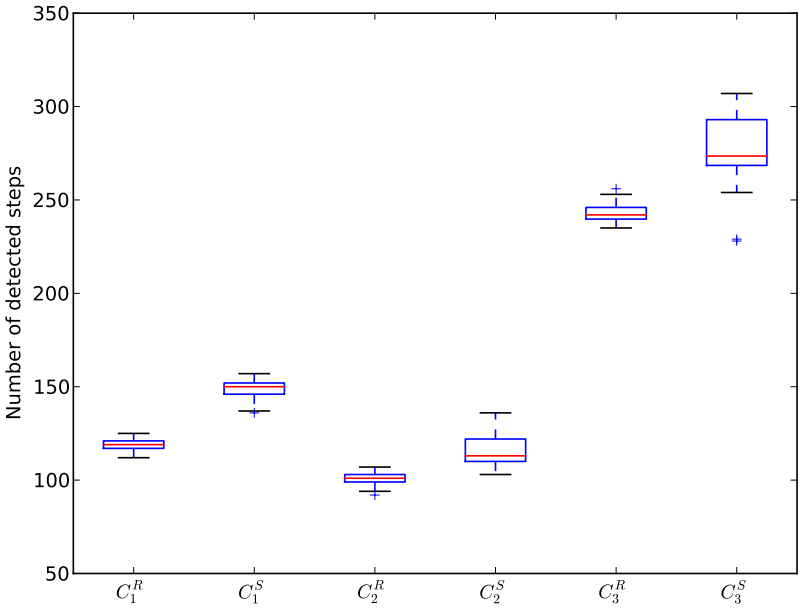


Figure 4.14.: Statistics of step counts

Table 4.1.: Statistics of stride lengths of each participant in m

	$\bar{x}$	s	$x_{\min}$	Q_1	Q_2	Q_3	$x_{\max}$
S	0.62	0.07	0.61	0.61	0.61	0.61	1.68
R	0.84	0.11	0.61	0.79	0.84	0.89	1.68

PDR algorithm resorted to using its minimum step angle threshold of 20°, yielding the observed stride length.

Figure 4.15 reinforces this hypothesis. Thereon, all detected stride lengths of R (top) and S (bottom) for route C_1 have been plotted. One can clearly identify the shift in stride length after the red line. It marks the beginning of the last three runs, which have been performed another day. On this day, while obviously wearing different clothes, the stride length estimation significantly improved for S, whereas it deteriorated for R. Unfortunately, these three runs were the only ones within all three routes with a proper stride length detection of S's steps. Luckily, R's steps after the red line are the exception and most are detected properly throughout all routes.

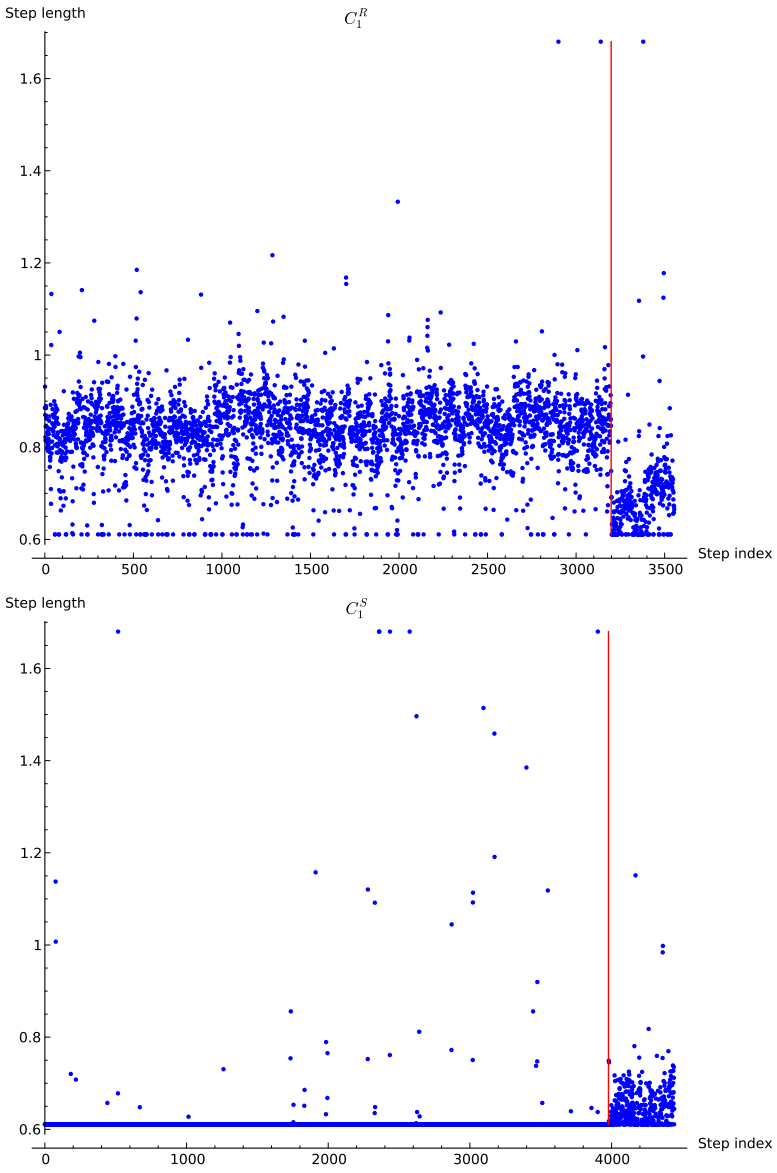
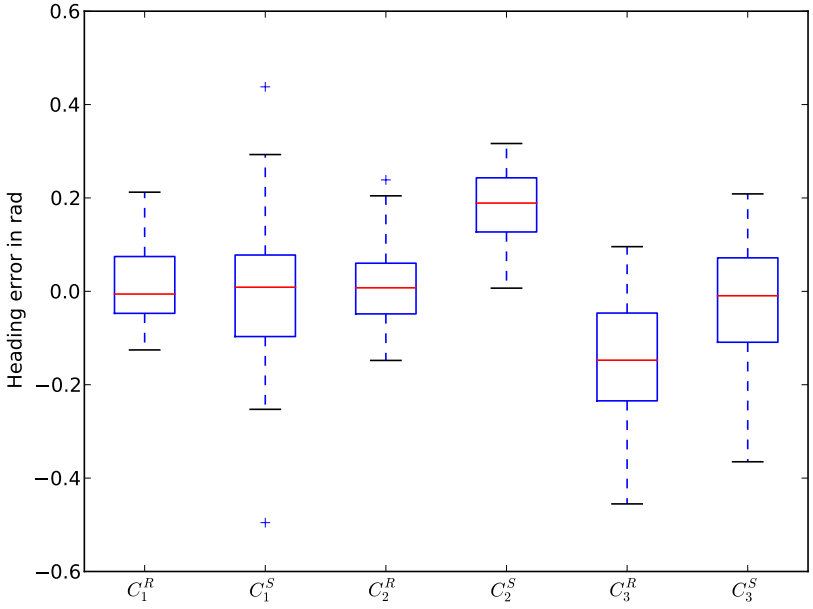


Figure 4.15.: Stride lengths for all steps detected on route C_1 per person, the vertical line marks the beginning of runs conducted another day

Regarding angular error (Fig. 4.16), not surprisingly, the results of R for C_2 and C_3 are similar to their single counterparts the same person walked. For route C_1, both participant's distribution of error is centered around zero, but S exhibits a larger spread and two outliers. Regarding route C_2, S's error distribution is more shifted towards positive values, meaning that the angles of the four-sided route are underestimated. It is worth noting, that both partic-



	l	$\bar{x}$	s	$x_{\min}$	Q_1	Q_2	Q_3	$x_{\max}$
C_1^R	97	0.02	0.09	-0.13	-0.05	-0.01	0.07	0.21
C_1^S	97	-0.01	0.17	-0.50	-0.10	0.01	0.08	0.44
C_2^R	84	0.02	0.09	-0.15	-0.05	0.01	0.06	0.24
C_2^S	84	0.18	0.08	0.01	0.13	0.19	0.24	0.32
C_3^R	197	-0.14	0.13	-0.46	-0.23	-0.15	-0.05	0.10
C_3^S	197	-0.04	0.16	-0.37	-0.11	-0.01	0.07	0.21

Figure 4.16.: Statistics of angular error in rad

ipants walked C_2 in different directions, R clockwise and S counter-clockwise, but wearing the phone both in their right pockets, possibly introducing gyroscope bias. On route C_3 on the other hand, the participants had their phone in a different pocket each, mostly facing each other in order to facilitate the audio-based proximity sensing.

4.2.2. Collaborative PDR Performance

Whereas section 4.2.1 evaluates each participants' PDR recordings individually, this section examines the results of their collaboration. In order to collaborate, the participants need not do more than passing each other (or sometimes stand still for a second). The audio-based proximity detection recognizes that they are close and initiates an exchange of their PDR estimations via Bluetooth.

As mentioned before, all experiments that did not yield an exchange have been discarded. These can all be tracked down to Bluetooth connectivity issues or other software factors, which is beyond the control of *reckonMe*. Table 4.2 shows how often a certain number of exchanges occurred per route.

Table 4.2.: Number of exchanges that occurred per route

	1	**2**	**3**
C_1	30	0	0
C_2	9	21	0
C_3	5	14	11

Like designed, on route C_1 the participants experienced one exchange per run. C_2 on the other hand, yielded the possible two exchanges in only two thirds of the cases, the reason is an interrupted Bluetooth connection. Upon starting, both participants stand close to other and a connection is established, but no exchange is performed, as they have not walked yet. After parting ways, the connection usually drops and sometimes fails to re-establish (Sec. 3.3) either on the first meeting or at the end.

On route C_3 however, the hindering factor was the noise generated in the pocket of R while steadily walking together. The threshold of at least 60 m having to be walked before exchanging (again), leaves only small windows of opportunity for the proximity to be detected three times. Furthermore, these windows are getting smaller or even disappear, should the proximity not be detected in a timely manner. A second exchange after an overall 150 m walked, would render a third exchange impossible as it had to take place after 210 m, which is longer than the route. Hence, only a third of the runs on route C_3 yielded three exchanges, consolingly on the other hand, only a sixth yielded only one exchange.

Considering the worrisome results of S described in section 4.2.1, one might wonder if these lead to a decline in collaborative PDR performance. Luckily, this is not the case.

The following analysis of collaborative PDR performance is confined to the absolute displacement at the end of a run, as the exchanges can neither correct for angular error nor for issues regarding step detection. Due to the design of the correction steps, the impact of the collaboration can of course be negative for one or even both participants.

Table 4.3 reflects the results from an individual perspective, by classifying each run for each participant as either an improvement – in case the total error at the end of a run is smaller with the exchange than it would have been without it – or as a deterioration if the collaboratively acquired position lies further from the truth.

Table 4.3.: Outcomes of exchanges per individual, regarding total error

	C_1^R	C_1^S	C_2^R	C_2^S	C_3^R	C_3^S	Σ
improvements	20	16	15	19	24	20	**114**
deteriorations	10	14	15	11	6	10	**66**

Interestingly, R benefited from exchanges more often than S for C_1 and C_3, whereas S took more benefit than R from C_2. In total, in roughly two thirds ($\frac{114}{180}$) of the cases, results have improved for the individual, whereas in the other third, a deterioration took place.

Nevertheless, it makes more sense to evaluate a collaborative system on a collective basis, that is regarding the two participants as part of the same system and hence consider the errors of both simultaneously for each run. This has been done in the following and is summed up in table 4.4, where the displacements of both participant with exchanges were added for each run and compared to the sum of their individual displacement, had there not been any exchanges.

Again, about two thirds of the runs benefited from the collaborative approach, whereas less than one third suffered from a deterioration. Not only do the deteriorations appear less frequently than the improvements, they are – on average – also smaller in size, as shown in figure 4.17. It is worth noting, that these numbers refer to the sum of error of both participants. While the improvements for routes C_1 and C_2 are only slightly bigger than the deteriorations, the improvements for route C_3 are huge compared to their deteriorations.

Table 4.4.: Outcomes of exchanges on a system level, regarding total error

	C_1	C_2	C_3	Σ
improvements	20	19	22	**61**
deteriorations	10	11	8	**29**

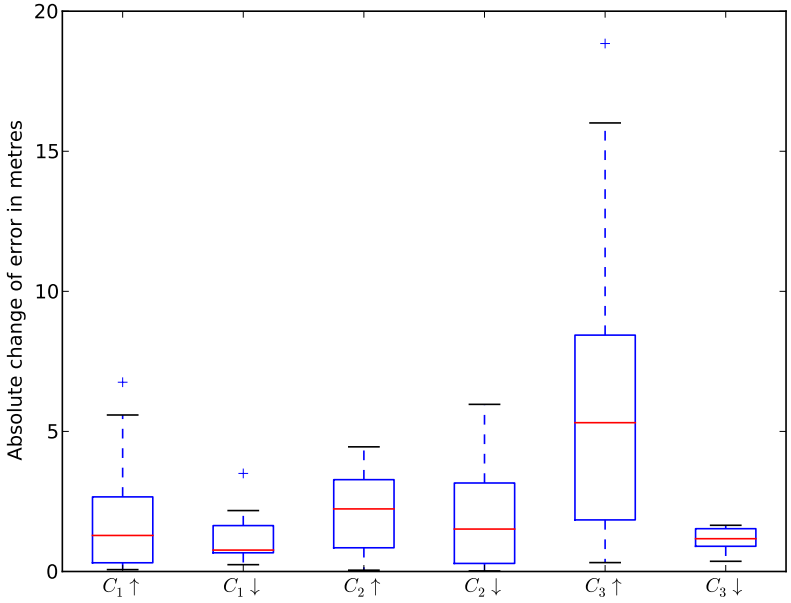
Figure 4.18 however, paints a clearer picture. It shows statistics of the relative change in error per run, that is, the fraction of the absolute improvement and the error without exchanges. With the exception of route C_2, the relative improvements are significantly higher than the deteriorations. In other words: the larger the error, the larger the improvement, compared to a relatively smaller deterioration.

4.2.2.1. Exchanges Exemplified

Figure 4.19 shows two examples of exchange on route C_1. It shows the traces of both participants, which begin at either end of the corridor and end in an arrow, the exchange "steps" are highlighted using bold red arrows. Please note, that the exchange steps join both traces at the same location, which can lead to confusion of the traces when following them with the eyes. In the run shown on the top, both participants drifted in the same direction and hence had a rather similar perception of where the were. As a result, the shift introduced by the exchange is small for both, as represented by relatively short red arrows. In this case, the exchange was neither beneficial, nor harmful for both participant's PDR estimates.

The bottom traces of figure 4.19 on the other hand, show a highly beneficial exchange. There, R drifted southwards, whereas S did not until the exchange but drifted northwards afterwards. Cancelling each other's drift out, let their traces end nearly perfectly where they started. It is worth noting that the correction steps are almost perpendicular to the path, which indicates that both participants "agreed" on where they were lengthwise along the path.

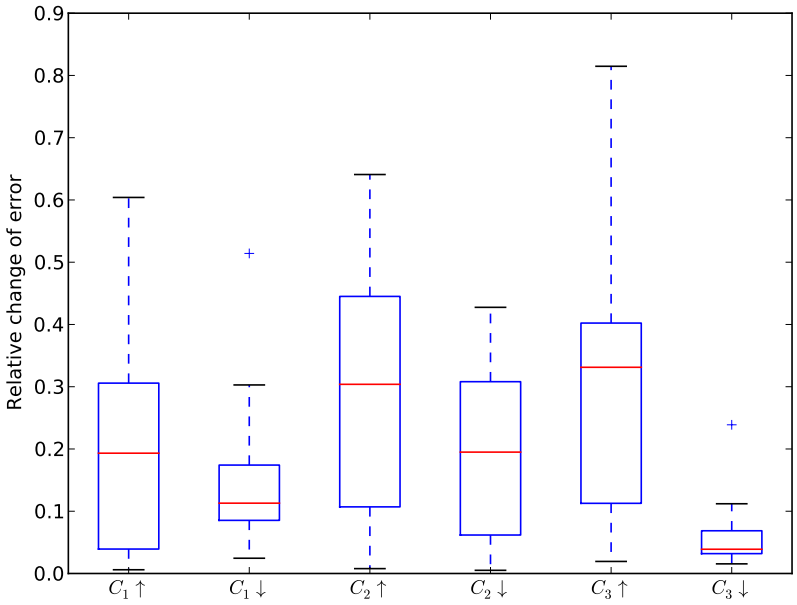
Figure 4.20 shows two runs on route C_2. One can clearly see the underestimated path length by S, as the traces are considerably smaller. This turns out to be a problem for the run shown at the top. Having only one exchange, it leaves the participants worse off, as the exchange shifts R's estimate further southwards, adding to its error, and shifts S's estimate northwards by almost the same amount as its error at the end, thereby introducing it.



	l	$\bar{x}$	s	$x_{\min}$	Q_1	Q_2	Q_3	$x_{\max}$
$C_1\uparrow$	97	1.86	1.94	0.07	0.31	1.29	2.66	6.76
$C_1\downarrow$	97	1.23	0.99	0.24	0.67	0.76	1.64	3.50
$C_2\uparrow$	84	2.19	1.45	0.05	0.85	2.23	3.28	4.45
$C_2\downarrow$	84	1.91	2.01	0.02	0.29	1.51	3.16	5.97
$C_3\uparrow$	197	5.80	4.93	0.32	1.84	5.31	8.44	18.85
$C_3\downarrow$	197	1.15	0.45	0.36	0.90	1.17	1.53	1.65

Figure 4.17.: Statistics of absolute improvements (↑) and deteriorations (↓) of displacement error through exchanges per run on a system level

The second run on the other hand, yielded two exchanges that nearly cancelled each other out, thereby diminishing the error introduced by the underestimation of S's stride lengths. The first exchange shifts R south-westwards and S in the opposite direction, whereas the second exchanges almost reverses theses shifts. It is worth noting that the second exchange occurred a few steps too early, as both PDR algorithms detected additional steps, shown by both arrows protruding beyond the exchange point. In this case, the exchanges were neither beneficial nor harmful, but at least "correcting themselves". The nine runs (Table 4.2) where only one exchange occurred most likely contributed to



	l	$\bar{x}$	s	$x_{\min}$	Q_1	Q_2	Q_3	$x_{\max}$
$C_1\uparrow$	97	0.21	0.18	0.01	0.04	0.19	0.31	0.60
$C_1\downarrow$	97	0.17	0.14	0.02	0.09	0.11	0.17	0.51
$C_2\uparrow$	84	0.29	0.20	0.01	0.11	0.30	0.45	0.64
$C_2\downarrow$	84	0.19	0.14	0.01	0.06	0.19	0.31	0.43
$C_3\uparrow$	197	0.31	0.22	0.02	0.11	0.33	0.40	0.81
$C_3\downarrow$	197	0.07	0.07	0.02	0.03	0.04	0.07	0.24

Figure 4.18.: Statistics of relative improvements (↑) and deteriorations (↓) of displacement error through exchanges per run on a system level

the deteriorations seen on C_2.

Figure 4.21 shows three runs on route C_3. The first run (above) had only one exchange, but even more would not have helped due to S's disastrous path. The second example on the other hand, features two exchanges and a fairly good result. While the first exchange, again, appears to mediate between the different perceptions of path length, the second exchange partly reverses that and elegantly averages out the drifting apart that each participant experiences.

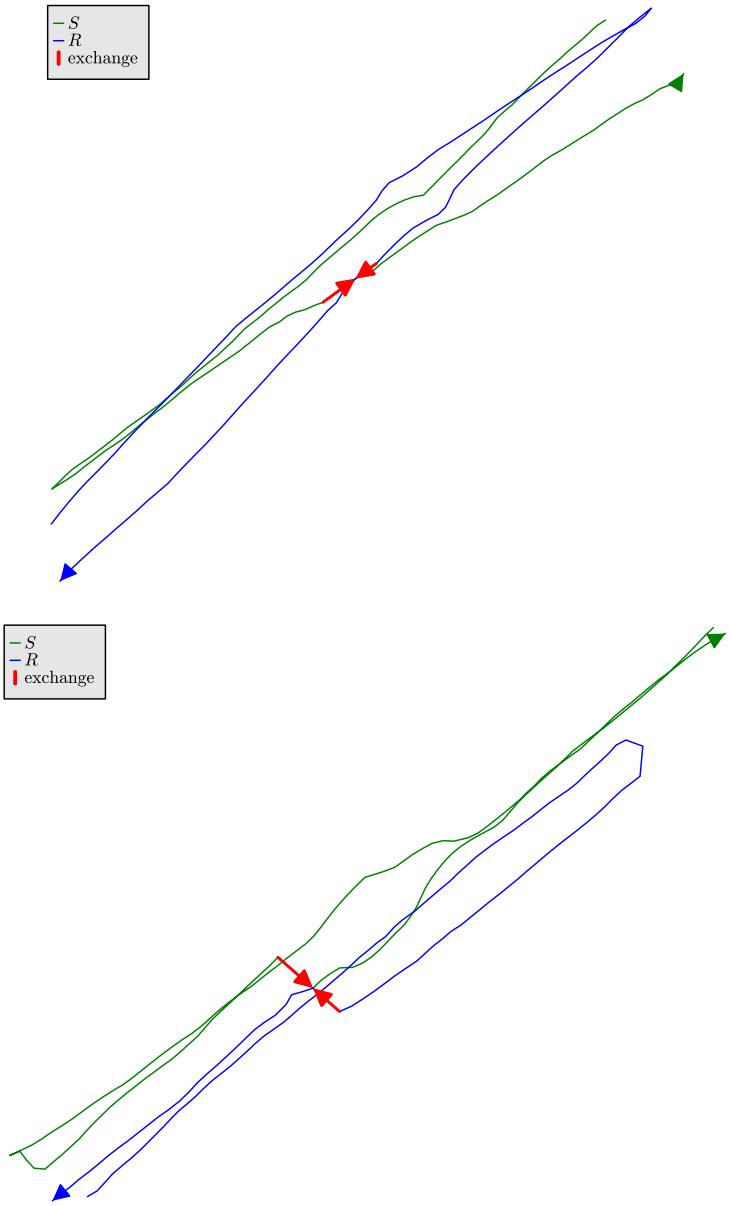


Figure 4.19.: Examples of exchanges for C_1

The third run shows three exchanges that also yielded a very good result. Where the first exchange does not appear to have much influence, as the participants are still quite close in their estimates, the second exchange mediates their path lengths as well as corrects a drifting apart. The third exchange mostly corrects for the different path lengths again and yields a quite accurate

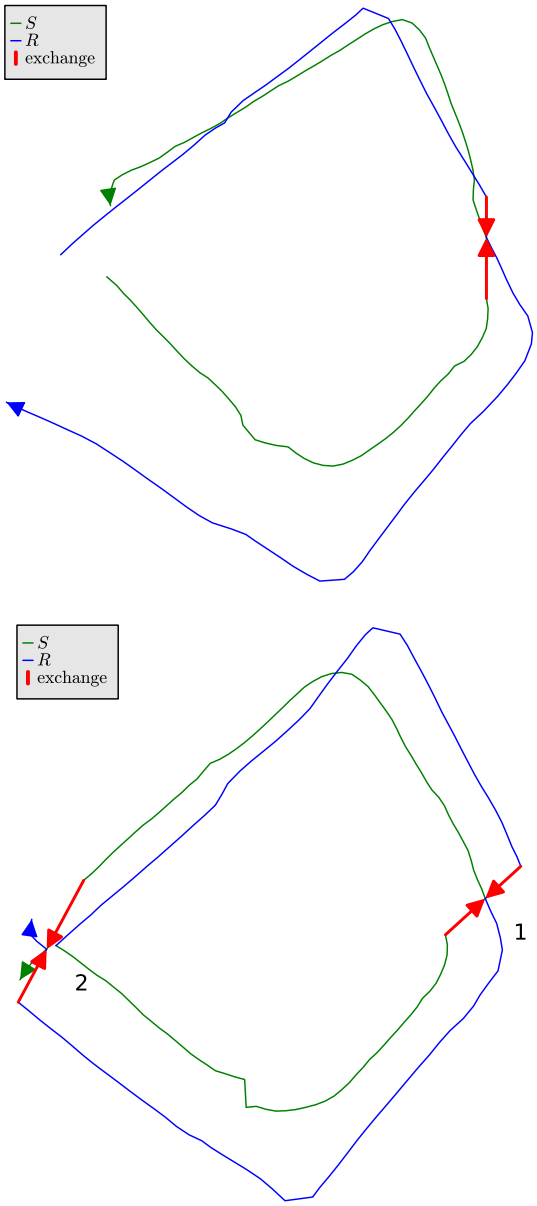


Figure 4.20.: Examples of exchanges for C_2

result.

It is noteworthy that all three typical accumulation points of exchanges on route C_3 are shown. The first and second run both feature the door that had to be opened, in the middle of the last straight segment. The second and third run feature the exchange that typically occurred when both partic-

ipants turned around (number 1 and 2, respectively). Additionally, the third exchange shows an exchange quite near the end, which is the last chance for a third exchange.

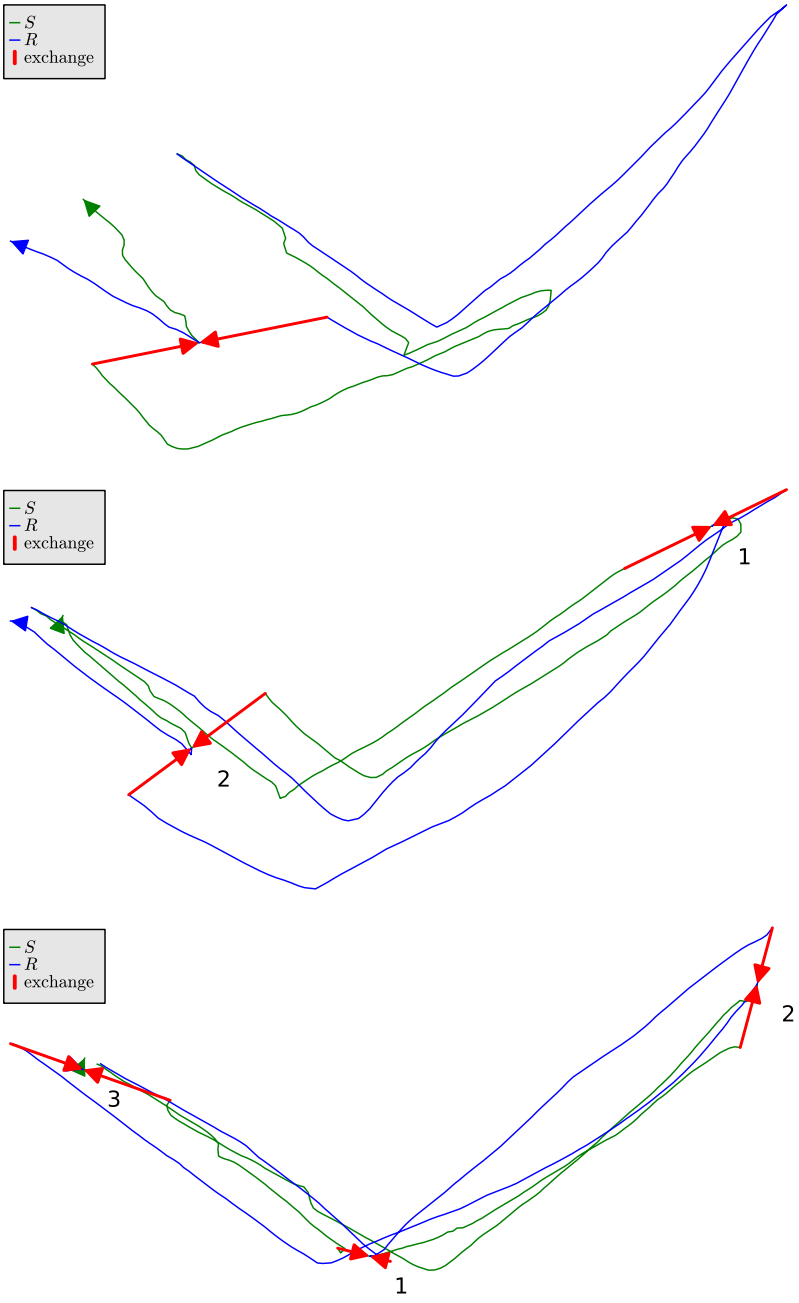
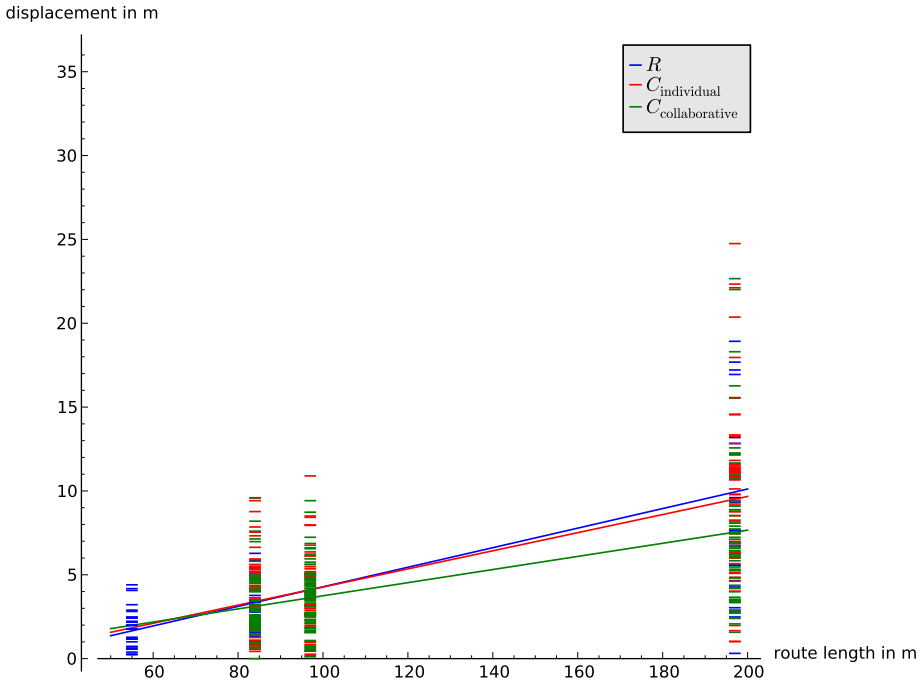


Figure 4.21.: Examples of exchanges for C_3

4.2.3. Individual vs. Collaborative PDR Performance

Section 4.1 shows the very promising results of the PDR algorithm applied to a single person. Section 4.2.1 on the other hand examines the PDR results of $N = 2$ participants on an individual basis.

Whereas the overall results match the ones aquired with only a single person, the results of the other participant (S) are somewhat worrisome. Due to a trouser pocket which was too "loose", a mis-detection of steps lead to much shorter paths. This more than anything represents the real-world nature of the collaborative PDR-approach, as one does not always find ideal conditions. Nevertheless, section 4.2 shows that this approach is resilient to such errors to a certain extent, as the improvements outnumber as well as outweigh the deteriorations.



R	0.05833
$C_{\text{individual}}$	0.05402
$C_{\text{collaborative}}$	0.03914

Figure 4.22.: Average displacement per meter of route length in m

Figure 4.22 shows a scatterplot of all 270 (90 single, $2 \cdot 90$ collaborative) absolute displacements and a fit for each group acquired through linear regression.

As expected, the slope of the single person R and each participant regarded individually $C_{\text{individual}}$ are nearly the same, with $C_{\text{individual}}$ even being a little smaller, likely due to the smaller paths of participant S.

Through collaboration however, an average displacement of 5.4 % of the path length could be reduced to 3.9 %, which is an improvement of 27.5 %.

4.3. Audio-based Proximity Detection

As mentioned in section 2.2.1, the following evaluation refers to the algorithm as published in [15]. Nevertheless, the results of the unaltered algorithm are still interesting to look at and – to a certain extent – apply here because the threshold values were left untouched:

Overall, a total of 137 minutes of audio data (thereof 64 minutes without emitting a signal for estimating the algorithm's precision) for the standing and walking scenarios as listed in table 4.5 and table 4.6 have been collected. The phones were carried in jeans pockets, with no extra cases. The parameters of the algorithm were chosen with an ambitious goal of delivering reliable results, with no time lag, high resolution (up to 2.5m), to be run live on the device. In particular, false positives (which according to our definition included also proximity detection of people standing more that 3m apart) were considered much more harmful than false negatives (i.e. not detecting proximity). Thus, at the price of a slightly lower recall rate, the bar for the precision of the algorithm was set very high.

With auto-correlation thresholds $t_L = 0.355$, $t_H = 0.42$, and 9 out of 14 d_p-values for a positive detection (translating to a time resolution of approximately one second), the true negative rate amounted to 99.95%, meaning a nearly perfect precision of the algorithm.

First, a social interaction scenario, i.e. a situation, when two people are standing next to each other was investigated. For this static case a majority-decision window of 10 seconds (which is not too long, considering nobody is moving) was used. The results are presented in table 4.5 and figure 4.23. Figure 4.23 shows the proximity detection rate as a function of distance between the two persons and the majority threshold cut-off for the 10-second recognition window.

Overall, the algorithm almost perfectly recognized proximity up to 2 m (both indoors & outdoors), while situations when people were more than 3 m apart were correctly classified as non-proximate.

Table 4.5.: Standing scenario (indoor & outdoor). Recall rates for 1 s and 10 s window (majority-decision). In a 10 s variant, distances up to 2 m are well recognized, proximity of 4 m and greater is considered being too far away, as desired.

Activity	Dist. [m]	Recall (1s)	Recall (10s)
Office corridor	0.6	0.95	1
Shopping mall escalator	1	0.71	1
Supermarket queue	1	0.53	0.60
Office corridor	1.2	0.94	1
Busy street crossing	1.2	0.80	1
Busy street crossing	2	0.22	0.25
Neighbouring rooms	2	0	0
Office corridor	2.5	0.38	0.31
Shop shelf in between	3	0.06	0.11
Busy street crossing	3	0.01	0
Office corridor	4	0.14	0
Busy street crossing	4	0	0
Busy street crossing	5	0	0
Office corridor	6	0.09	0
Supermarket	6	0	0
Office corridor	10	0	0

Next, the walking scenario was investigated. In this case, when the receiver starts walking, the signal-to-noise ratio (SNR) drops from 9.5 dB to 2 dB, thus making it extremely hard to cover both, standing and walking, with a fixed set of parameters. This can also be seen in figure 4.24, which depicts the average detection rate of 11 recordings, increasing every second as the sender approaches the standing receiver, bearing a peak of nearly 100% when closest and more rapidly falling than rising due to acoustic shielding of the leaving person's body. With the receiver walking, on the other hand, the detection rate both, increases at a slower rate and the duration of subsequent detections is significantly lower.

Results for the walking scenario are shown in table 4.6. As anticipated, there is a big discrepancy in the recall rates between the "standing receiver" and the

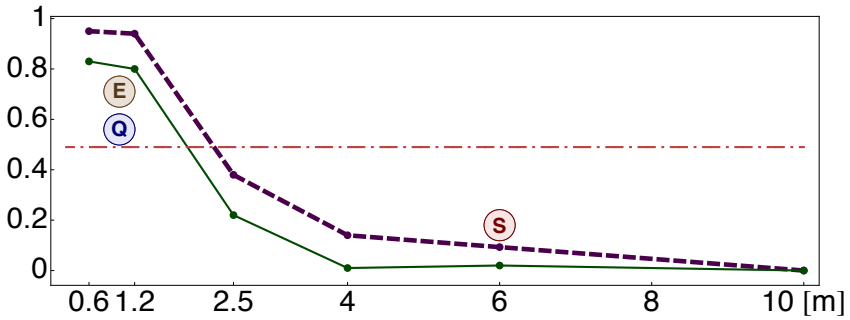


Figure 4.23.: Overall recall rates as a function of distance with both persons standing: indoors (dashed), near a busy street crossing (solid), (E) on an escalator, (Q) in a supermarket queue, (S) remotely at different supermarket shelves. Hortizontal line corresponds to the majority-decision window of 10 seconds (see Table 4.5).

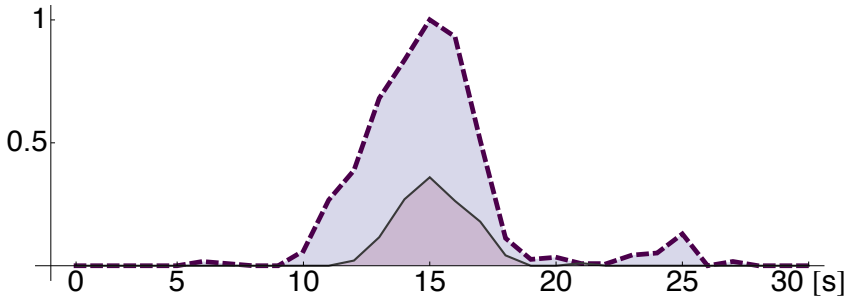


Figure 4.24.: Average detection rates with one person standing: (a) sender walking past receiver (dashed) and (b) receiver walking past sender.

"walking receiver" scenario. One of the reasons of the 0.33 recall rate for the walking receiver is a very high bar on the precision rate of the algorithm.

Table 4.6.: Walking in a sports shop: sender walking towards and away of the receiver (11 recordings), receiver walking towards and away of the sender (9 recordings).

Activity	Precision	Recall
Receiver Standing	1	1
Receiver Walking	1	0.33

4.3.1. Alteration for reckonMe

In order to increase the algorithm's performance for the scenario at hand, which is walking persons, the focus has been shifted away from a successful detection of persons standing. Instead, the top-most majority decision layer (9 out of 14) has been removed and exchange of PDR positions is initiated once a detection candidate occurs. Furthermore, this increases the time resolution from one second to $l_p \approx 0,07$s (Sec. 2.2).

While this makes the algorithm overly sensitive to people standing several meters apart, this does not pose a problem for the conceived routes, as the participants are either constantly in motion or kept from exchanging by setting the threshold of minimum meters to be walked to an appropriate value.

Although this new approach has not been tested as vigorously, its successful application shown in section 4.2.2 serves as a proof of concept. It is worth noting, that no false positives occurred and its recall rate reached 1 by using a small "trick". The trick was, that the participants stopped near each other for a second on routes C_1 and C_2, in case an exchange did not already occur while approaching each other.

The runs with no second exchange on route C_2 (see table 4.2), can clearly be traced to Bluetooth connectivity problems, as indicated by a grayed-out Bluetooth icon on the devices while performing the experiments. On route C_3, the trick was not applied deliberately, but rather occurred naturally when the participants turned or had to open a door. Nevertheless, exchanges also occurred while walking, which of course required a successful detection.

5. Conclusion

The aim of this work has been to create a collaborative pedestrian dead-reckoning system that runs on commodity hardware without the need for infrastructure. The resulting system was implemented as standalone iPhone app that performs all operations in real time, on the device. On the one hand, this proves the feasibility of such an approach. On the other hand this shows that fusing various sensing systems such as gyroscope and sound can lead to greater knowledge.

The results (Sec. 4.2.3) show that the collaboration of two persons in PDR systems is possible and beneficial, even if one of said persons achieves less than optimal results with dead-reckoning. The displacement error, growing with distance, could be reduced from 5.4 % to 3.9 % on average, which is an improvement by 27.5 %.

While the author pointed out flaws in Apple's documentation of the iOS-platform, this is the exception rather than the norm. Although being closed-source and derogatorily called a "walled garden", the platform is a pleasure to work with. From an engineering perspective, nothing grave has been missed as the APIs are well-designed and work as advertised. The comparability of sensor data due to uniform hardware further adds to this impression.

Future work in this area will certainly benefit from advances in the device's sensing capabilities and their accessibility through APIs. One of these advances is the adoption of Bluetooth 4.0 Low Energy, enabling low power appliances like position beacons, that, with the help of properly designed APIs enable an even finer-grained proximity detection. A concrete implementation is the announced iBeacon functionality of iOS 7, which is still subject to a non-disclosure agreement (NDA).

Even a small improvement in APIs, such as informing the app of a correction of the device's orientation reference frame when using a magnetometer-corrected references frame, would greatly help the PDR system's heading accuracy. Using sound as a proximity proxy on the other hand, is inherently device-independent and could be used across operating system borders, supported by a device-independent means of radio communication.

Bibliography

[1] W. A. Arentz and U. Bandara. Near ultrasonic directional data transfer for modern smartphones. In *Proceedings of the 13th international conference on Ubiquitous computing*, UbiComp '11, pages 481–482, New York, NY, USA, 2011. ACM.

[2] L. Girod, M. Lukac, V. Trifa, and D. Estrin. The design and implementation of a self-calibrating distributed acoustic sensing platform. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 71–84, New York, NY, USA, 2006. ACM.

[3] Y. Jin, H.-S. Toh, W.-S. Soh, and W.-C. Wong. A robust dead-reckoning pedestrian tracking system with low cost sensors. In *Pervasive Computing and Communications (PerCom), 2011 IEEE International Conference on*, pages 222–230, 2011.

[4] K. Kloch, P. Lukowicz, and C. Fischer. Collaborative PDR Localisation with Mobile Phones. In *Proceedings of the 2011 15th Annual International Symposium on Wearable Computers*, ISWC '11, pages 37–40, Washington, DC, USA, 2011. IEEE Computer Society.

[5] K. Kloch, G. Pirkl, P. Lukowicz, and C. Fischer. Emergent behaviour in collaborative indoor localisation: an example of self-organisation in ubiquitous sensing systems. In *Proceedings of the 24th international conference on Architecture of computing systems*, ARCS'11, pages 207–218, Berlin, Heidelberg, 2011. Springer-Verlag.

[6] E. Martin, O. Vinyals, G. Friedland, and R. Bajcsy. Precise indoor localization using smart phones. In *Proceedings of the international conference on Multimedia*, MM '10, pages 787–790, New York, NY, USA, 2010. ACM.

[7] L. M. Ni, Y. Liu, Y. C. Lau, and A. P. Patil. Landmarc: indoor location sensing using active rfid. *Wirel. Netw.*, 10(6):701–710, Nov. 2004.

[8] L. Ojeda and J. Borenstein. Personal dead-reckoning system for gps-denied environments. In *Safety, Security and Rescue Robotics, 2007. SSRR 2007. IEEE International Workshop on*, pages 1–6, 2007.

[9] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan. Beepbeep: a high accuracy acoustic ranging system using cots mobile devices. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 1–14, New York, NY, USA, 2007. ACM.

[10] N. B. Priyantha. *The cricket indoor location system*. PhD thesis, Massachusetts Institute of Technology, 2005.

[11] J. Qiu, D. Chu, X. Meng, and T. Moscibroda. On the feasibility of real-time phone-to-phone 3d localization. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 190–203, New York, NY, USA, 2011. ACM.

[12] M. Schaff. Indoor navigation made easy path recognition pdr traces using android smartphones. Master's thesis, University of Passau, March 2013.

[13] J. Scott and B. Dragovic. Audio location: accurate low-cost location sensing. In *Proceedings of the Third international conference on Pervasive Computing*, PERVASIVE'05, pages 1–18, Berlin, Heidelberg, 2005. Springer-Verlag.

[14] U. Steinhoff and B. Schiele. Dead reckoning from the pocket - an experimental study. In *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, pages 162–170, 2010.

[15] B. Thiel, K. Kloch, and P. Lukowicz. Sound-based proximity detection with mobile phones. In *Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones*, PhoneSense '12, pages 4:1–4:4, New York, NY, USA, 2012. ACM.

[16] M. Wirz, D. Roggen, and G. Tröster. A wearable, ambient sound-based approach for infrastructureless fuzzy proximity estimation. In *Proceedings of the 14th IEEE International Symposium on Wearable Computers (ISWC 2010)*. IEEE Computer Society, Oct. 2010.

A. DVD-ROM contents

The DVD contains the source code, all the recorded data and evaluation scripts used. It has the following directory structure:

Source code contains the Xcode project with the source of the iPhone app

Audio data contains all audio data recorded for the sound-based proximity evaluation

Audio evaluation scripts contains the Mathematica scripts used for the sound-based proximity evaluation

PDR data contains all recordings of the analyzed traces

PDR evaluation scripts contains all the scripts used for evaluating the PDR performance. For this purpose, the open-source mathematics software *Sage* has been used, which can be downloaded for all major operating systems at http://www.sagemath.org/

Erklärung der Urheberschaft

Ich erkläre hiermit, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum Unterschrift