Radboud University

Master degree thesis in Artificial Intelligence

Human SLAM

Simultaneous Localisation and Configuration (SLAC) of indoor Wireless Sensor Networks and their users

August 11, 2015

Supervisors:

A.C. (Anne) van Rossum MSc.
Almende B.V. & DoBots B.V.
Rotterdam

Dr. W.F.G. (Pim) Haselager
Radboud University
Nijmegen

Second assessor:

Dr. I.G. (Ida) Sprinkhuizen-Kuyper
Radboud University
Nijmegen

Author:

Wouter Bulten BSc.
Student number: s3040585

"I may not have gone where I intended to go, but I think I have ended up where I needed to be."

Douglas Adams, *The Long Dark Tea-Time of the Soul*, 1988

Abstract

The indoor localisation problem, in smart spaces, is more than just finding the whereabouts of users. Finding positions of users *relative* to the devices of a smart space is even more important. Unfortunately, configuring such systems manually is a tedious process, requires expert knowledge and is not resilient to changes in the environment.

We propose a new system, called *Simultaneous Localisation and Configuration* (*SLAC*), to address these two challenges, locating the user and the devices, and combine them into a single estimation problem. The SLAC algorithm, which is based on FastSLAM, is able to locate devices using the received signal strength indicator (RSSI) of devices and motion data from users.

Simulations have been used to show two main effects on the localisation performance: the amount of RSSI updates and the location of devices in the space. Live tests, in non-trivial environments, showed that we can achieve room level accuracy and that the localisation can be performed in real time. This is all done locally, i.e. running on a user's device, with respect for privacy and without using any prior information of the environment or device locations.

More work is required to increase accuracy in larger environments and to make the algorithm more robust for environment noise caused by walls and other objects. Existing techniques, such as map fusing, could alleviate these problems.

Preface

This thesis was written as a completion of a Master's degree in Artificial Intelligence at the Radboud University. As part of this project I did an internship at Almende and DoBots, two research-focussed companies located in Rotterdam.

The problem of indoor localisation was first presented to me by Anne van Rossum, who was my external supervisor. At the very early stages of the project Anne said to me: "Smart buildings are the hype, but it is of importance to make them truly smart". This set the tone of the project: we need intelligent solutions with a good theoretical basis. My (re)gained knowledge in the field of mathematics should be mainly credited to Anne. I want to thank Anne for his guidance during the project.

In April of last year I asked Pim Haselager, my internal supervisor, whether he had any ideas about interesting companies for my internship. That first email was the start of this project. From that point on Pim's most important contribution to this project was the thing he does best: asking difficult questions. "How would we apply this to...? How is this relevant for...?" It wasn't uncommon for Pim to compare the project to the localisation mechanism of owls or even fictional nobleman such as *Baron Munchausen*. I want to thank Pim for all these questions and support. His statement "your thesis should hurt a little" is something I will probably never forget.

Besides my two supervisors there is a long list of people to thank, which includes: everyone at Almende and DoBots who gave feedback and helped with the live tests, Panagiotis Chatzichristodoulou for finding a very critical bug in my code and the numerous discussions, my parents for their continuous support at every stage, Robert-Jan Drenth for our frequent update talks, Ida Sprinkhuizen-Kuyper for accepting to be the second assessor, Rotterdam Community Solutions for granting me access to their office, the porter of the Spinoza building for letting me roam the basement on a Saturday, Bart Dekker for proofreading and everyone with whom I discussed my project and gave feedback. Without their support this project wouldn't be where it is now.

Wouter Bulten
August 11, 2015

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

With advances in electronics and computer science, technology has become an indispensable part of our daily lives. A new field, where intelligent system have not (yet) been fully integrated, is the work and living environment: our homes and offices. Although these spaces are full of intelligent devices, there is often no link or collaboration between intelligent devices and the environment we live in. The current development of the *Internet of Things* (Atzori et al., 2010), which attempts to connect devices and, by doing so, creating *smart spaces*, is about to change that. By connecting individual devices and sensors we can build systems that as a whole interact with a user.

A key problem (or challenge) within these smart spaces is indoor localisation: making estimates of users' whereabouts. Without such information, systems are unable to react on the presence of users or, sometimes even more important, their absence. This can range from simply turning the lights on when someone enters a room to customising the way devices interact with a specific user.

Even more important for a system to know where users exactly are, is to know where users are *relative* to the devices it can control or use to sense the environment. This relation between user and device location is an essential input to these systems. A central question in this field is therefore:

> **What are the locations of devices in a smart space and what are the current locations of users relative to these devices?**

In this thesis we propose a new system, called *SLAC*: *Simultaneous Localisation and Configuration*, to address the two challenges, locating the user and the devices, and combine this into a single estimation problem. With SLAC we aim to *simultaneously* locate both the user and the devices of a system deployed in an indoor environment. To accomplish this goal we use characteristics that are already available in many smart spaces: signal strength measurements (or *RSSI*) from devices and motion data from smart phones and other portable device (see also Figure 1.0.1). We will combine these two inputs in a system that can locate users and devices, respect individual users' privacy and perform all estimations in real time. SLAC is based on a common technique from robotics, *simultaneous localisation and mapping (SLAM)*, and in particular the *FastSLAM* algorithm (Montemerlo et al., 2002).

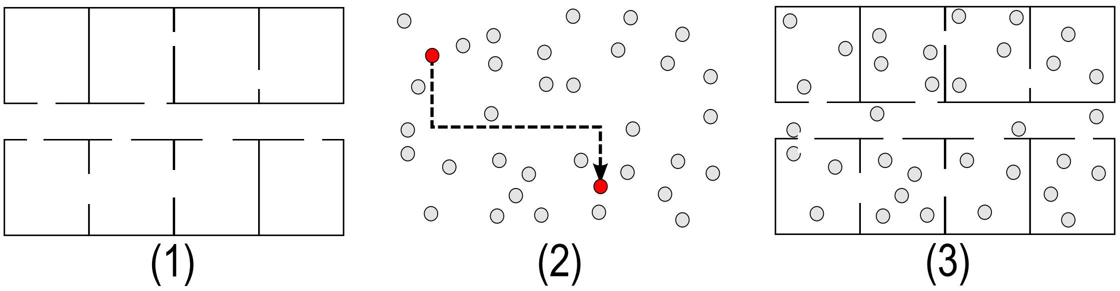


Figure 1.0.1: General project overview. In a building (1) a set of devices (grey dots) is installed (2) with a user (red dot) that walks around (dashed line). Based on measured RSSI values and motion data from the user we try to learn the location of devices in the building (3).

1.1 Potential & challenges of smart spaces

Smart spaces become more effective when there is a smooth interaction between users and devices in a well-mapped environment. For homes these systems can result in more comfort for residents and increase energy efficiency (Han and Lim, 2010; Batista et al., 2013). For healthcare it can result in more tailored care, an increase in self-reliance and a decrease in social isolation for patients and elderly (Chan et al., 2008; Stankovic et al., 2005; Skubic et al., 2009). In a work environment, a smart office can improve work efficiency and job satisfaction (Röcker, 2010) and reduce energy consumption (Choi et al., 2015). In other words, in the full spectrum of our living environment, these smart spaces can have great benefits. Not only for end users (residents, patients, etc.) but also for organisations and other owners of such facilities. Especially in times where healthcare costs continue to rise and energy consumption should be minimised, these systems can offer a (partial) solution to these problems. In these cases we do however need to tread carefully as to not infringe on users' privacy.

Regardless of this potential, many challenges still exist which can slow down the adoption of these systems in our environments. Users of smart spaces will, in general, be non-experts that lack the technical skill to install and connect these complex systems. This is not a great problem for large scale applications where specialists can install and maintain systems. However, to make these systems available to general consumers the installation, configuration and management of such systems should be easy. This is one of the main issues that prevents our homes from becoming intelligent: it is too difficult to set up and connect devices to create a collaborative system. Moreover, after these systems have been set up the work is not done. Devices, including sensors, can be moved by users to accommodate for changes in the environment or their condition. And even if installation is carried out by specialists, due to this dynamic nature of our environments, systems should adapt to changes themselves to prevent tedious (re)configuration steps and prevent faults.

A second, not to be underestimated problem, is that of simply controlling devices themselves. Elderly, patients in hospitals, children, all can have a reduced ability to control or interact with devices or smart spaces in general. For office and home environments the focus is on ease of use and efficiency. Direct and explicit interactions with the smart space should be minimised to reduce cognitive load.

1.2 Self-* properties & ProHeal

Given the challenges of deploying a smart space, in an ideal world, a smart space should be a self-managing system. It should automatically adapt to changes, configure itself and run autonomously. More precisely, the system should support a list of *self-** properties (Sterrit and Bustard, 2003a,b; Warriach et al., 2014) which have the goal to minimise human intervention. These properties range from automatic configuration of the systems to adapting to changes in the environment and the presence of users.

Such a self-managing system introduces an interesting trade-off: from the systems' perspective there is a need for information and user input. Generally, the more (and the better) the user input, the higher the performance of the system. From the perspective of the user the opposite is true: especially in terms of enjoyment and ease of use, we want to minimise the load requested by the system.

The concept of a self-managing system is key in the *ProHeal* project[1] which is funded by the *Information Technology for European Advancement (ITEA2)* research programme and is part of the *Ageing society & wellbeing challenge*. ProHeal runs from January 2014 till July 2016. Through the DevLab[2] consortium, Almende participates in this project. The goal of the ProHeal project is as follows (Gijssel and Stam, 2014):

> "Autonomic systems have so-called self-managing properties, like self-configuration, self-optimization, and in particular self-protection and self-healing. There is an increasing demand on these self-managing properties for many software systems deployed in dynamically changing environments, such as smart buildings, healthcare systems, disaster management, etc. Such systems must be able to easily adapt at runtime in response to changes in their user preferences, requirements, computing infrastructure and surrounding physical environment. For this reason, these systems must be flexible, fault-tolerant, configurable and secure.
>
> [..] The expected impact will be a significant reduction in development and maintenance costs of systems and service management organisations as well as a mitigation of risks associated with shutting down and restarting the system for adapting it. Moreover, since the systems-to-be will adapt autonomously, user satisfaction and experience will be greatly improved."
>
> *Retrieved from the ProHeal Project Outline Annex, p.4*

1.3 Indoor localisation

In this thesis we focus on a subset of self-* properties in the context of indoor localisation. More specific, we try to minimise the time users need to invest in configuration and managing the network of devices in smart spaces.

When a system knows the location of a user it can react on its presence (or on its absence). When the current location can be computed a system can, generally, also derive where the user has been and make assumptions (interpolate) where the user will be in the future. So, in the field of smart spaces, a central question is:

[1]`https://itea3.org/project/pro-heal.html`
[2]`https://www.devlab.nl/`

What is the current location of a user?

In outdoor environments localisation of users is straightforward by using systems such as GPS. GPS results in an absolute position of a user: regardless of the user, each GPS device can compute its location in a globally consistent coordinate system (e.g. using latitude, longitude and elevation). In indoor environments localisation using GPS (or comparable measures) is often inaccurate or not feasible as signals have difficulties penetrating walls of buildings and the resolution is too low.

Fortunately, for most systems in spaces, an absolute and global coordinate system is not required. The position of a user expressed in latitude and longitude is not particular useful, we are more interested in the room or section of a building the user is in. In other words, a system needs to know what the location of a user is *relative* to the environment and the devices therein. In many applications a rough measure of “close to”, with precision in meters, is sufficient. This relative localisation does however introduce a second question:

What are the locations of the devices that the system can use or control?

By estimating both the locations of the devices of the system and users we can compute distances, determine whether two ‘things’[3] are close and track paths of users.

Note that this implies more than just the distance between a user and a single device. Relative to the building we want to derive a map on which we can locate users and devices simultaneously. A simple scenario highlights this difference: consider a user entering a hallway. Here we do not only want to turn on the lights close to the user, but all (and only) the lights in the whole hallway.

In a smart space, devices can have various roles; e.g. an actuator (controlling something or being an output device, e.g. a lamp), a sensor (measuring some environment value, e.g. light intensity) or a combination of both. These roles are, however, not of direct importance for the localisation algorithm. So, from now on we do not focus anymore on the role of a specific device. We assume that there is a set of devices for which it is required that we have a location estimate.

1.4 Requirements for indoor localisation

Besides accuracy, i.e. minimising the localisation error, there are also additional requirements for an indoor localisation system. While there are many of these requirements, we will discuss three which are, in our opinion, the most important.

First of all, localisation should be preferably *online* and should give real time results; i.e. the position of a user must be computed in real time and updated when a user moves. Systems that derive users’ paths afterwards in an offline phase are generally not suitable for smart spaces as a system cannot directly act on a change of user’s location. Localisation of devices in a smart space does not necessarily share this requirement as their positions will change less frequently (e.g. a light or a fridge will not move every day).

A second requirement is that of *privacy*. There are many possible approaches for tracking a user through a building but there is a large risk of infringing on users’ privacy,

[3]This can be a user and a device but also two components or two users.

especially when these methods are used in (semi) public spaces such as hospitals or government buildings. A system that is not privacy-aware can hinder the adoption of such a system by end users. The risk is highest when using cameras or the localisation algorithm is centralised and users' locations are tracked and stored in a central system. In an ideal system we would only use input that is privacy-aware and perform the localisation decentralised. By, for example, performing localisation locally on users' devices we can give them more control over their data.

The last constraint is linked to the use of *additional hardware*. Localisation of users can be done quite reliably when additional sensors are used such as camera's, sonar sensors or specific devices that users have to carry. Also, the mapping of buildings is feasible using (expensive) laser scanners. However, to make sure that a localisation system is easily deployable to existing buildings and usable by users we favour a solution that requires as little additional hardware as possible. We can, for example, utilise characteristics from the devices we want to locate and use portable devices from users; e.g. almost all users will carry a mobile phone which, if used, removes the need for a application-dependent device that the user needs to keep.

1.5 Overview & Research Questions

In this thesis we address the problem of indoor localisation and focus especially on finding the location of devices within smart spaces. We attempt to answer the following main research question:

> **Is it possible, in principle, to design a fully autonomous self-localisation system for a network of devices by utilising the users of the space that the network is deployed in? And, if so, what level of accuracy can we achieve, measured in meters?**

In addition to this main research question we will address three subquestions:

1. To which degree can we satisfy our three main requirements, online localisation, respecting privacy and using no additional hardware, and how does this influence the overall accuracy of the system?
2. To what extent does the added information, generated by moving users, remove the need for prior location information, i.e. can we achieve full autonomous self-localisation without prior information or configuration regarding the location of devices?
3. How does the localisation error (in meters) of the system depend on the location of devices in the space?

In the next chapter we will first review related work regarding indoor localisation and describe the environment and sources of input we can use. Then, in Chapter 3 we will introduce the SLAC algorithm with which we try to answer our main research question. SLAC builds upon the *FastSLAM* robotics localisation algorithm which base, for the unacquainted reader is explained in Appendix A. The SLAC algorithm is privacy-aware and is an online localisation method; i.e. localisation starts whenever a user starts moving inside a building. Moreover, we focus on a solution that can be deployed in smart spaces without additional hardware requirements.

We will then explain the specific implementation and the simulations and live tests that have been carried out to asses the performance of our system in Chapter 4. The results of these tests are explained in Chapter 5. We conclude this thesis with a discussion of our research questions and the results in Chapter 6. We will show that room-level accuracy is indeed possible and that localisation of devices can be done very fast while fulfilling our requirements of privacy, online computation and using no additional hardware.

Additionally, four appendices are added to this thesis. Appendix A shows an overview of the techniques that form the basis of SLAC, including particle filters, (extended) Kalman filters and FastSLAM. In Appendix B we show a general overview of some mathematical basics that have been used in our thesis, especially regarding random variables and distributions.

At the start of this project a pilot was conducted using a different technique than eventually used in the SLAC algorithm. In Appendix C and D we explain the theory behind these techniques, which are *Gaussian Processes* and *Gaussian Process Latent Variable Models*. After the pilot we favoured the FastSLAM approach; the rationale behind this decision is further explained in the next chapter.

Chapter 2

Background & Related work

Indoor localisation algorithms that try to locate system components and users are usually related to some form of a *Wireless Sensor Network (WSN)*. In its most basic version, a WSN is a group of sensors (often called 'nodes') that can communicate with each other or with some other entity wirelessly. If we consider every device in a smart space to be a node we can apply a broad range of techniques and literature focussing on WSNs to our indoor localisation problem.

In this chapter we will give an overview of different types of localisation methods regarding WSNs. While it is infeasible to show all possible directions an attempt has been made to give a broad overview of the directions one can take when trying to locate nodes inside a network.

2.1 Range & Environment

Nodes in a WSN communicate with each other and with external systems through some wireless system such as *ZigBee*, *Bluetooth* or *WiFi*. This communication layer comes with a 'free' input that is often used in localisation: the *received signal strength indicator* (RSSI). The RSSI value resembles the power of a received radio signal (measured in *dBm*). The higher the RSSI value, the higher the signal strength.

The rationale behind using RSSI values is that almost all wireless systems report and use this value natively; i.e. no additional sensors are required to measure RSSI values. It can therefore be considered as a free input to a system. Moreover, and this is specifically interesting for localisation, there is a relation between RSSI and distance which can be roughly described using some form of the *Log-distance path loss model* (Seidel and Rappaport, 1992; Patri and Rath, 2013):

$$\text{RSSI} = -10n\log_{10}(\frac{d}{d_0}) + A \qquad (2.1.1)$$

with d the relative distance between transceiver and recipient, n the signal propagation exponent and A_0 a referenced RSSI value at d_0. Usually d_0 is taken to be 1 such that A_0 becomes the single strength measured at a distance of 1 meter of the node.

Following Equation 2.1.1, in a ideal world, the RSSI value is only dependent on the distance between the two nodes. In reality, however, RSSI values are heavily influenced by the environment and have, consequently, high levels of noise. This noise is, for example, caused by multi-path reflections: signals bounce against objects in the environment

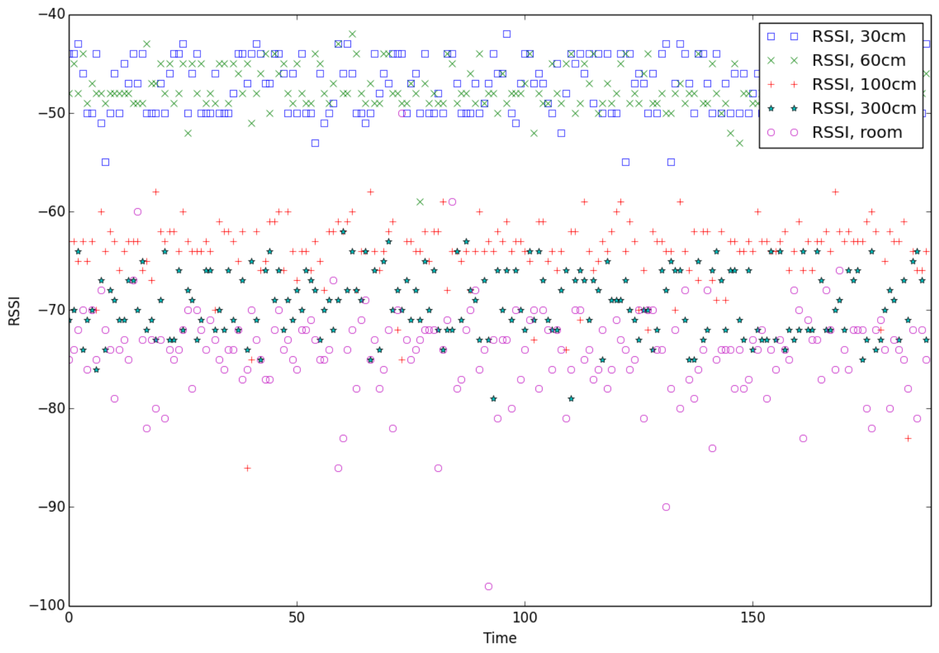


Figure 2.1.1: RSSI measurements over time. The received signal strength of a device is clearly influenced by distance but the amount of noise is substantial. For this plot, a bluetooth device was set up as a beacon to continuously broadcast its unique identifier. Another bluetooth device was placed at various distances from the beacon and acted as a recording device. With a 1Hz sample rate RSSI values were sampled. For the 'room' case, the beacon was placed in an adjacent room to show the effect of walls.

such as walls. An example of the effect of noise and distance on RSSI values is shown in Figure 2.1.1.

2.2 Localisation algorithms

The literature regarding localisation and WSNs is broad. A convenient way to categorise existing indoor localisation techniques is by examining what exactly they attempt to locate. Roughly three approaches can be distinguished: *1*) locating nodes; *2*) locating users; *3*) and localising both.

2.2.1 Locating nodes

Localisation algorithms that focus on localising nodes can utilise so called *anchor nodes*; algorithms that do not use any anchors are termed *anchor-free*.

Anchor-node driven algorithms assume that a subset of nodes from the network have knowledge about their true position. By using these anchor nodes as a base the other nodes are 'anchored' on the global coordinate frame and it is often possible to position nodes absolutely. The anchor-based localisation problem of devices in a WSN can be formally described as:

Anchor-based localisation problem: Given a network N consisting of nodes n. Each n_i has a coordinate vector $c_i \in \mathbb{R}^d$ with $d \in \{2, 3\}$ which

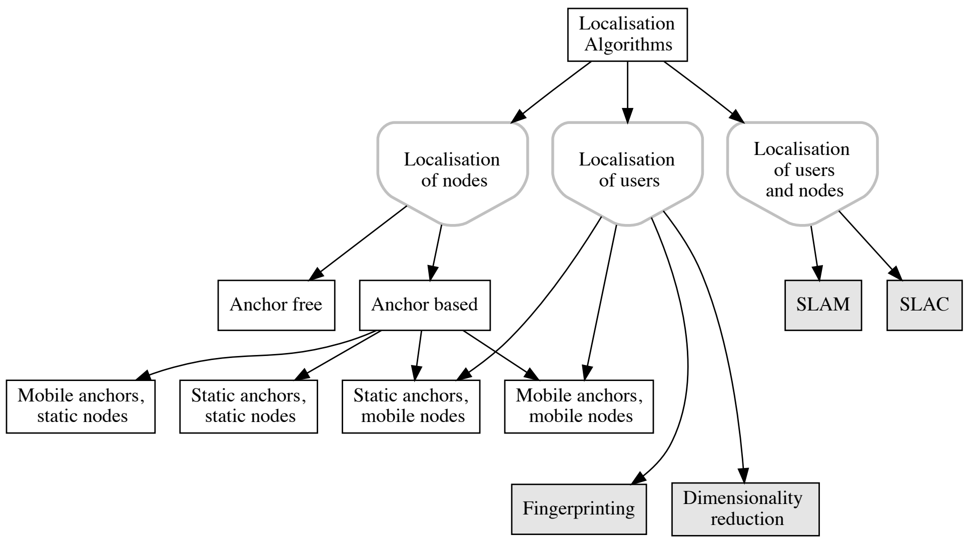

Localisation Algorithms
Localisation of nodes
Localisation of users
Localisation of users and nodes
Anchor free
Anchor based
SLAM
SLAC
Mobile anchors, static nodes
Static anchors, static nodes
Static anchors, mobile nodes
Mobile anchors, mobile nodes
Fingerprinting
Dimensionality reduction

Figure 2.2.1: Overview of localisation algorithms. The most basic division is based on what the algorithms try to localise: users, nodes or both. There can be some overlap between the categories when users are considered as moving nodes. Light grey boxes are example localisation techniques for that specific category. *SLAC* is an example of an algorithm that tries to locate both users and nodes simultaneously.

are unknown to the node themselves. A subset $L \subset N$ does know their coordinates c_i and are considered *anchors*.

What are the coordinates of all nodes $n_i \in (N - L)$ given anchors L?

The main benefit of anchor-based localisation is that, given enough anchors, the problem is solvable. Anchor based localisation algorithms roughly come in four categories (Han et al., 2013):

1. *Static anchors, static nodes:* The most simple setup, all nodes are static. Examples of this are Ekberg and Ngai (2011) who use swarm localisation, Li and Kunz (2007) who use a version of non-linear dimensionality reduction and the particle swarm optimisation approach by Chuang and Wu (2008).

2. *Static anchors, mobile nodes:* Anchor nodes are static and do not move and mobile nodes try to find their location. See for example Rabbat and Nowak (2004). These algorithms can also be used to track humans when they are considered a moving node.

3. *Mobile anchors, static nodes:* Mobile anchors are often in environments where many anchors are infeasible or the distance between nodes is high. E.g. a single mobile anchor equipped with GPS can localise many static nodes by driving around (Sreenath et al., 2007).

4. *Mobiel anchors, mobile nodes:* All nodes are mobile, e.g. applicable to robot swarms where only a few robots know their true positions. See for example the work

by Sheu et al. (2010) or the Monte-Carlo localisation approach by Hu and Evans (2004) or Baggio and Langendoen (2008) who show that mobility can improve performance.

By using anchors a reliable estimate of device locations can be made but this requires us to have prior information about the network. Given our attempt to eliminate all prior information, anchor-based approaches are not useful for solving our indoor localisation problem.

Anchor-free localisation algorithms come closer to our desired solution. These algorithms do not assume that nodes contain any information about their true position. This is particularly useful for situations where exact location information is infeasible; e.g. in remote areas or because of cost restrictions. Instead, anchor-free localisation algorithms use measurements to find a consistent (relative) map; e.g. using triangulation. The problem can be defined as:

> **Anchor-free localisation problem:** Given a network N consisting of nodes n. Each n_i has a coordinate vector $c_i \in \mathbb{R}^d$ with $d \in \{2, 3\}$ which are unknown to the node themselves.
>
> What are the coordinates of all nodes $n_i \in N$?

The problem with anchor-free localisation is that it is almost impossible to do this localisation without using additional sensors or input. There are many factors that influence the (measured) signal strength[1] and without directional information it is impossible to deduce the true network topology.

In other words, if we do not want te resort to using anchor nodes some other input is required. Here users of smart spaces come into play.

2.2.2 Locating users

Apart from algorithms that primarily focus on localising nodes, there are many examples of algorithms that focus solely on locating users or some other mobile entity. When a WSN is used for the localisation nodes are often used more implicitly and less as active participants of the algorithm; for example by defining a unique fingerprint of an environment.

The localisation of users can be defined in two ways: online and offline. In the offline method a users' path is derived at the end. The online method continuously updates the estimate of the users' position.

> **Localisation of users:** Given a user u whose location at time t is described by a vector $l_t = \{x_t, y_t\}$.
>
> *Offline localisation:* What are the locations of the user between $t = 0$ and $t = N$?
> *Online localisation:* Given the users previous positions $\mathbf{l}_{0:N}$ what is the current position l_{N+1}?

An often used and also intuitive method to localise users is fingerprinting. At runtime or before deployment a 'fingerprint' of the environment is created, this can for example

[1]E.g. nodes can be perceived as being far away while in reality a wall is blocking the signal.

be a signal strength map. When a user needs to be located, the user matches its current fingerprint with a stored fingerprint database and can retrieve its location. Creating fingerprints of the building can be crowdsourced to reduce human intervention (Yang et al., 2012).

Ferris et al. (2007) were the first to use *Gaussian Process Latent Variable Models* (*GP-LVM*) for localisation which is an offline example of user localisation method. A combination of a motion model and a signal strength map of WiFi access points was used to reconstruct a user's trace. GP-LVM uses dimensionality reduction to map a higher dimensional space (the signal strength map) to a lower dimensional space (a x, y-map of the user trace). GP-LVM has also been used to locate robots in an indoor environment (Hollinger and Djugash, 2008).

The GP-LVM based localisation technique uses no additional sensors and can be used with existing hardware. This solution came close to matching all of our requirements. We therefore carried out a pilot study to investigate the benefits and possible drawbacks of this technique. An elaborate description and discussion can be found in Appendix D. Eventually we opted to not use GP-LVM for SLAC as its main benefit, a relatively good accuracy without prior information, did not outweigh two important drawbacks: the method only supports offline estimation and has, in its base version, no functionality for locating devices. Though, these estimates of device locations are a valuable input for systems in smart spaces. This last problem has been addressed by Hollinger et al. (2011) but they use additional hardware to be able to make predictions of node positions which increases the cost of using such a system and cannot be deployed without changing the environment (by adding additional sensors).

2.2.3 Locating users & nodes simulteanously

The third and last category, and the one we are interested in primarily, is that of localising users and nodes simultaneously. We already described that locating devices without additional information is hard. By combining information from both devices and users this can potentially be overcome.

The double localisation problem of locating both devices and users can be defined (again in two versions: offline and online) as:

> **Double localisation problem:** Given a user u whose location at time t is described by a vector $l_t = \{x_t, y_t\}$ and a network N consisting of nodes n with coordinates $c_n \in \mathbb{R}^d$ with $d \in \{2, 3\}$ which are unknown to the nodes themselves.
>
> *Offline localisation:* What are the locations of the user between $t = 0$ and $t = N$ and the locations of all nodes $n \in N$?
> *Online localisation:* Given the users previous positions $l_{0:N}$ what is the current position of the user t_{N+1} and the current estimate of all nodes $n \in N$?

There are multiple approaches to the double localisation problem, one of them using GP-LVM (Hollinger et al., 2011) as we described before.

Another field where this double localisation problem is an active topic of research is the field of robotics. Robots that are deployed in unknown environments have to simultaneously locate themselves, and map the environment in which they move. This is called the *Simultaneous localisation and mapping* (or in short *SLAM*) problem.

The estimation of the map and robot position is based on sensor readings and the *controls* (i.e. actions) the robot performed. The combination of both sensor readings and controls is important as the world is non-deterministic and the same control can result in a different result. The difficulty of SLAM is mainly caused by a problem which, at first sight, looks like a chicken-and-egg problem: a map is required for estimating a position, but for the mapping an estimate of the robot pose is required. SLAM overcomes this problem by estimating both at the same time.

SLAM is often centred around the observation of landmarks: distinguishable objects or markers in an environment that a robot can observe. These landmarks can be implicit, like walls, or more explicit such as (sensor) nodes and devices. Especially because of the use of devices as landmarks, SLAM can be used to locate the positions of devices in a WSN. Examples are the WiFI GraphSLAM algorithm of Huang et al. (2011) who use WiFi access points nodes or Menegatti et al. (2010) who use cameras to further refine node location estimations.

As the problem of SLAM fits our problem of indoor localisation nicely we opted to use a version of SLAM as the base of our algorithm. While there are many versions of SLAM algorithms, SLAC builds upon FastSLAM (Montemerlo et al., 2002) which is based on *particle filters*. FastSLAM is an online localisation method and has already been applied to WSNs. For example, Sun et al. (2009) use range measurements to map sensors in an environment. In the next chapter we will further look at the specific use of FastSLAM.

Chapter 3

SLAC: Simultaneous localisation and configuration

The SLAC system proposed in thesis builds upon a well-known technique from robotics: *Simultaneous localisation and mapping* (or in short *SLAM*). The SLAM problem is defined as follows:

> Given a robot's controls and sensor readings what is the current estimated location of the robot and map of the environment?

The key difficulty of this problem is that a map is required for estimating a position while for the mapping an estimate of the robot pose is required; SLAM solves this by estimating both at the same time.

Until now we used the general term 'device' to indicate some object of our smart space that we want to locate. In our application there is also another type of device: the device carried by the user. To make a clear distinction between the two we will, from now on, use the general term 'landmark' for the devices we want to locate. This term is often used in the SLAM literature for a feature of the environment that is used to build the map.

While there are many versions of SLAM, SLAC builds upon FastSLAM (Montemerlo et al., 2002) which is an online SLAM algorithm that uses *particle filters* (and *Rao-Blackwellized particle filters* in particular) to do the state estimation. Individual devices, i.e. the landmarks, are represented by *extended Kalman filters (EKF).*

In this chapter we focus on the definition of the SLAC algorithm and how we map the robotics problem to the domain of indoor localisation. Notation wise we follow the definitions as defined by Thrun, Burgard, and Fox (2005). A basic understanding of particle filters and extend Kalman filters is assumed. For the uninformed reader, Appendix A contains a general overview of the theory behind these techniques.

We start with a description of the input and which transformations have to be applied before these source of input can be used. Then we will explain each individual component of the system, ranging from estimating positions to finding locations of devices. Last we will address the time complexity of the algorithm and look at a few specific situations, including devices that are moved.

3.1 Source of input

FastSLAM requires two types of input: measurements or predictions of motion to make pose estimations and environment data to resample particles and to build the map (in our case, finding locations of devices). In our application, signal strength (RSSI) measurements are used to determine distances to devices. Pose estimations are made using *inertial measurement unit (IMU)* data. Before these measurements can be used as inputs they first need to be preprocessed.

3.1.1 RSSI Filtering

As shown in the previous chapters, especially in Figure 2.1.1, the raw RSSI signal contains noise. To filter out large spikes while trying to retain distance information a (regular) Kalman Filter is used to filter incoming signal strength measurements. We assume static landmarks to simplify the filter. The true RSSI value (without noise) is defined as the state we want to estimate. Our transition and observation model can then be reduced to:

$$\begin{aligned} x_t &= A_t x_{t-1} + B_t u_t + \epsilon_t \\ &= x_{t-1} + \epsilon_t \end{aligned} \tag{3.1.1}$$

$$\begin{aligned} z_t &= C_t x_t + \delta_t \\ &= x_t + \delta_t \end{aligned} \tag{3.1.2}$$

where ϵ_t and δ_t describe Gaussian noise and A_t, B_t and C_t our transition models. A_t and C_t are set to identity matrices as we assume the state is static (i.e. $x_t = x_{t-1}$) and we directly model the state (i.e. we assume $x_t = z_t$). Because there is no control, B_t is set to zero. The prediction step of the Kalman filter then becomes:

$$\bar{\mu}_t = \mu_{t-1} \tag{3.1.3}$$

$$\bar{\Sigma}_t = \Sigma_{t-1} + R_t \tag{3.1.4}$$

where R_t is the process noise and is typically set to a small value (e.g. 0.008). Because we model RSSI directly our measurement vector is a scalar value set to 1, this gives us the following reduced Kalman gain:

$$K_t = \bar{\Sigma}_t(\bar{\Sigma}_t Q_t)^{-1} \tag{3.1.5}$$

The measurement noise Q_t is set to the variance of the RSSI measurements. The state can then be updated using the Kalman gain:

$$\mu_t = \bar{\mu}_t + K_t(z_t - \bar{\mu}_t) \tag{3.1.6}$$

$$\Sigma_t = \bar{\Sigma}_t - (K_t \bar{\Sigma}_t) \tag{3.1.7}$$

The result of the Kalman filter on a sample of raw RSSI data can be seen in Figure 3.1.1. The Kalman filter is able to remove a large part of the noise from the data, but as a tradeoff, has to give up a bit of the responsiveness.

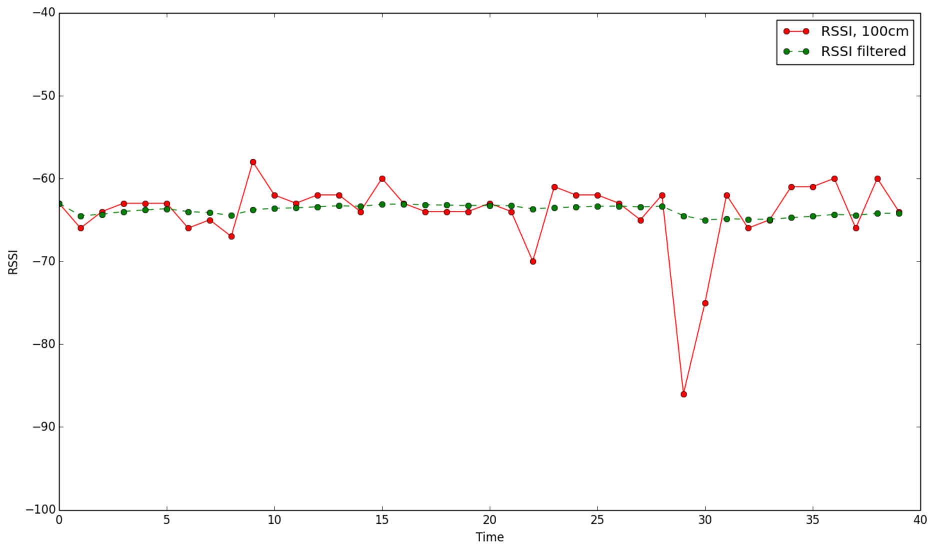


Figure 3.1.1: The effect of a Kalman filter on raw RSSI data sampled from a static device (i.e. no movement at both the receiver or transmitter end). The Kalman filter removes a large part of the noise from the signal.

3.1.2 Motion measurements

The second input is focussed on modelling motion. Two sensors are used to measure the motion of users: an accelerometer and a compass. These two sensors are present in almost any modern mobile device (including phones, tablets and wearables).

The compass returns the current rotation or heading relative to the global north; this does not require any processing and can be used directly as an input. Accelerometers return acceleration in three axes, x, y, z, and need to be processed to make estimations about the distance that is travelled.

The acceleration is used as the input for a pedometer. The pedometer counts steps which can then be converted to distance. Our pedometer is based on a design by Zhao (2010) and Ménigot (2014) and uses a sliding window of the acceleration data to determine whether a step has been made.

First, to make the pedometer rotation independent, the norm of the acceleration vector is computed. This makes sure that it does not matter how the devices is orientated (which can differ a lot, e.g. the differences between holding a tablet and a phone inside a pocket). Noise is removed using a regular Kalman filter. A new step is detected if the data satisfies the follow constraints (see also Algorithm 3.1):

1. The difference between the maximum and minimum acceleration in the time window must exceed the sensitivity. The sensitivity is defined as double the normalised acceleration variance.

2. The current acceleration must exceed the average acceleration in the time window. The previous acceleration must be below this average to ensure a single step is not counted twice.

3. The previous slot in the time window should not be a step. This ensures that a

Algorithm 3.1 Single step of the pedometer. Input is the acceleration in x, y and z (in m/s^2), current time step t, a sliding window V containing the norms of the acceleration of the previous steps and a binary array S keeping track of previous steps. The function returns 1 if a step has been detected and 0 otherwise.

```
function PEDOMETER($x, y, z, t, V, S$)

    Calculate norm $||v|| = \sqrt{x^2 + y^2 + z^2}$
    Filter $||v||$ using Kalman filter
    Remove first value of $V$, add $||v||$ to $V$

    $acc_{max} = \max(V)$
    $acc_{min} = \min(V)$
    $acc_{\text{threshold}} = \frac{acc_{max} + acc_{min}}{2}$
    Compute sensitivity $acc_s$

    if $(acc_{max} - acc_{min}) > acc_s$ then
        if $V_t \geq acc_{\text{threshold}}$ and $V_{t-1} < acc_{\text{threshold}}$ then
            if $S_{t-1} == 0$ then
                $S = S + [1]$
                return $\{1, S, V\}$                    ▷ Step detected
            end if
        end if
    end if
    $S = S + [0]$
    return $\{0, S, V\}$                                ▷ No step detected
end function
```

step cannot directly be follow by a next step.

3.2 Mapping the SLAM problem to indoor localisation

With our input defined (the RSSI measurements and motion estimates) we can map the SLAM problem to the domain of sensor networks and indoor localisation. The robot's controls, which are used for pose sampling, are replaced by our motion estimates. The observations, which are often 2D[1] measurements, are replaced by 1D RSSI measurements similar to the approach of Sun et al. (2009). Using the FastSLAM algorithm (See Appendix A, Algorithm A.3) as a base, the mapping results in the system described by Algorithm 3.2 and Figure 3.2.1. In the following subsections each individual component is described in detail.

3.2.1 Pose sampling

The flow and update rate of the SLAC algorithm is controlled by the pedometer: the algorithm is run after a new step has been detected. The step size (the distance a user moves after taking a single step) can be taken as a fixed value or derived from the

[1]When directional range sensors are used angle information of the observations can be derived.

Algorithm 3.2 Single step of the SLAC algorithm. Input is the particle set of the previous step Y_{t-1}, the current motion estimate u_t, all RSSI measurements since the previous step Z_t and the number of particles M.

```
1:  function SLAC($Y_{t-1}, u_t, Z_t, M$)
2:      $\bar{Y}_t = Y_t = \emptyset$

3:      for $m = 1$ to $M$ do
4:          Retrieve $\{x_{t-1}^{[m]}, L, w_t^{[m]}\}$ from $Y_{t-1}$      ▷ $L$ is a set of landmark EKF's
5:          Sample $x_t^{[m]} \sim p(x_t | x_{t-1}^{[m]}, u_t)$      ▷ Sample step
6:          $w_t^{[m]} = w_{t-1}^{[m]}$
7:          $\bar{Y}_t = \bar{Y}_t + \{x_t^{[m]}, L, w_t^{[m]}\}$
8:      end for

9:      for $i = 1$ to $|Z_t|$ do

10:         Identify correspondence $j$      ▷ Correspondence is trivial given device IDs
11:         Retrieve $z_t^i$ from $Z_t$

12:         if $j$ is not initialised then
13:             Update initialisation filter for $j$
14:             Compute $\Sigma_{j,t}$ of estimate

15:             if $\Sigma_{j,t} \leq$ threshold then
16:                 for m = 1 to M do
17:                     Initialise EKF with $\mu_{j,t}^{[m]}$ and covariance $\Sigma_{j,t}^{[m]}$
18:                 end for
19:             end if
20:         else
21:             for m = 1 to M do      ▷ Landmark update step
22:                 Update mean $\mu_{j,t}^{[m]}$ and covariance $\Sigma_{j,t}^{[m]}$
23:                 $w_t^{[m]} = w_t^{[m]} f(z_t^i | \mu_{j,t}^{[m]}, \sigma_z)$      ▷ Importance factor, see Equation 3.2.17
24:             end for
25:         end if
26:     end for

27:     Compute $\hat{N}_{\text{eff}}$      ▷ Number of effective particles
28:     if $\hat{N}_{\text{eff}} \leq N_{\text{threshold}}$ then      ▷ $N_{\text{threshold}}$ defines global threshold for resampling
29:         for m = 1 to M do      ▷ Resampling step
30:             draw $k$ from $\bar{Y}_t$ with probability $\propto w_t^{[k]}$
31:             $Y_t = Y_t + \{x_t^{[k]}, L = \{\{\mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}\}, \ldots, \{\mu_{N,t}^{[k]}, \Sigma_{N,t}^{[k]}\}\}, w_t^{[k]} = 1\}$
32:         end for
33:     else
34:         $Y_t = \bar{Y}_t$
35:     end if

36:     return $Y_t$
37: end function
```

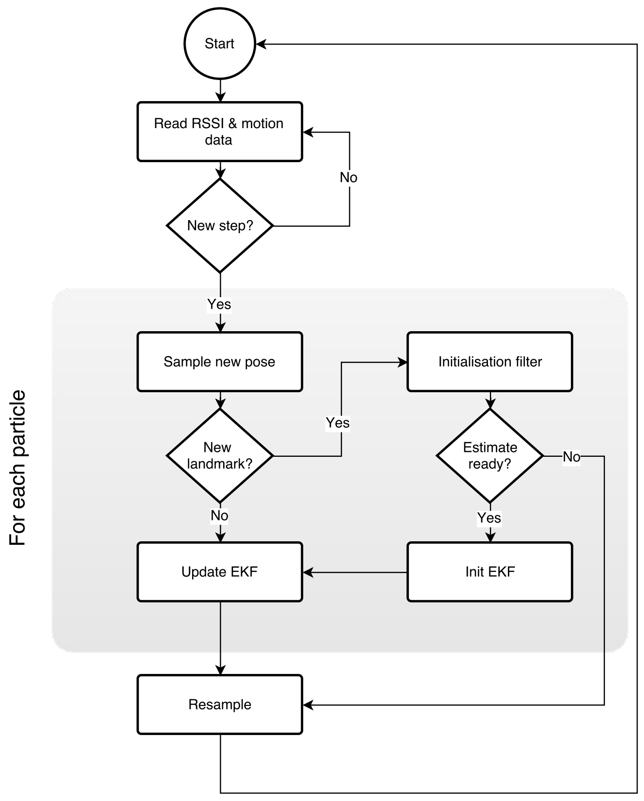


Figure 3.2.1: Visualisation of a single step of the SLAC algorithm. The diagram shows the update step of a single landmark based on a single observations. In reality, multiple observations and landmarks can be updated between the sample and resample step.

human's body height[2].

Given the step count[3], the prediction of the step size (r) and the current heading (θ) (taken from the compass) a new pose is sampled for each particle m:

$$\bar{\theta} = \mathcal{N}(\theta, \sigma_\theta) \tag{3.2.1}$$

$$\bar{r} = \mathcal{N}(r, \sigma_r) \tag{3.2.2}$$

$$x_t^{[m]} = x_{t-1}^{[m]} + [\bar{r}\cos(\bar{\theta}), \bar{r}\sin(\bar{\theta})] \tag{3.2.3}$$

where σ_θ describes the variance or noise of the compass readings and σ_r the variance of the estimated walking distance.

3.2.2 Initialisation using Particle Filters

Given the sampled pose of each particle we can start estimating the locations of each landmark (i.e. device) individually. As landmark observations are 1D and signals prop-

[2]Usually a value between 0.3 and 0.5 times the body height is used. The most common values are 0.413 for woman and 0.415 for males.

[3]In our simulations and live tests, the algorithm runs faster than the average time between steps. We can therefore compute a whole update after each step which results in a constant step count of one. However, this is not a requirement and multiple steps can be taken into account but this can lower the accuracy of the motion sampling.

agate spherical it is impossible, given a single measurement, to determine the bearing of an observation. We therefore must first make an initial estimate of a beacon location before we can refine it using the default method of FastSLAM.

An often used approach to overcome this initialisation problem is to divide the environment into a grid and use a voting scheme to find the most probable cell in this grid (Olson et al., 2006; Sun et al., 2009). Applications of this approach usually focus on robots (which have better motion modelling) or on relative small environments. For our indoor localisation we did not want to rasterise the environment so opted for a different approach using particle filters.

Given a range measurement z with variance σ_z we know that our landmark is somewhere on a circle with a radius equal to this measurement and a bandwidth proportional to σ_z. We can then create a particle filter with N particles which reside on this circle, with our current estimate of the user's position as its centre. The distance and angle (relative to the user) for each particle $i \in N$ are defined by:

$$d_i = \mathcal{N}(z, \sigma_z) \tag{3.2.4}$$

$$\theta_i = \frac{2\pi i}{N} \tag{3.2.5}$$

Distance (d_i) and angle (θ_i) are then converted to cartesian coordinates.

We now have a particle filter that can roughly estimate the beacon location. After each new measurement we update our filter by computing the importance weight and subsequently resampling the filter (using a low variance resampling). A particle's weight is updated using the probability density function (f) of the normal distribution:

$$w_{i,t} = w_{i,t-1} f(z | d_{i,u}, \sigma_z) \tag{3.2.6}$$

where $d_{i,u}$ is the distance between the user and the particle's estimate of the landmark (i.e. the expected measurement). How the particle filter converges to a beacon's location is depicted in Figure 3.2.2. When the variance between particles is low enough (given some threshold), we assume that a beacon's location has been found.

3.2.3 Refining using Extended Kalman Filters

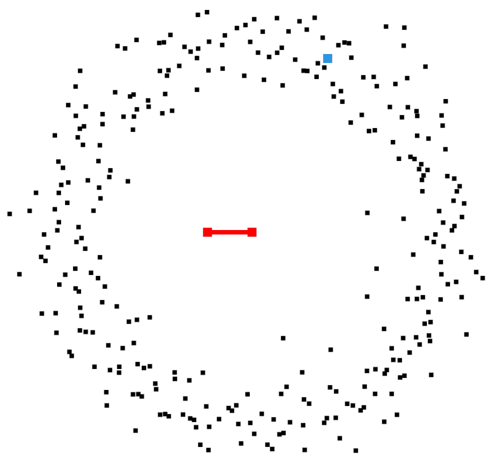
After the initialisation filter has converged we want to further refine the estimate. Our initialisation filter is a separate component that uses the current best user estimate as input. So, to improve on our rough initial estimate we move the estimation from the global initialisation filter to each individual particle. As it is impractical to update $M \times N$ particle filters (one particle filter per landmark per particle) we use an EKF to estimate a landmark's location (similar to the original FastSLAM implementation).

The state that our EKF tries to estimate is a 2D position vector; i.e. the x and y coordinates of the landmark. Our observations are however range only and 1D, this results in a slightly different EKF implementation. Our EKF implementation is based on the implementation by Sun et al. (2009) who use it in a range-only robot navigation problem.

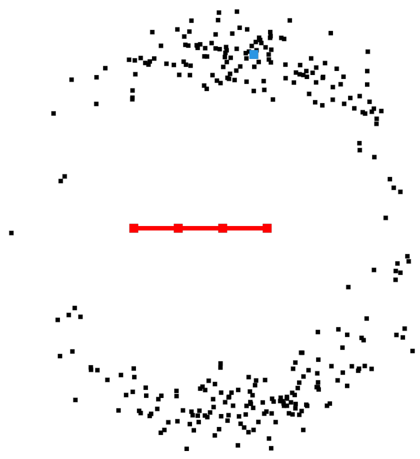
To initialise the EKF we use the estimate from the initialisation filter. The initial covariance matrix of the EKF of a landmark j is defined by the variance of the estimate:

$$\Sigma_{j,0}^{[m]} = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \tag{3.2.7}$$

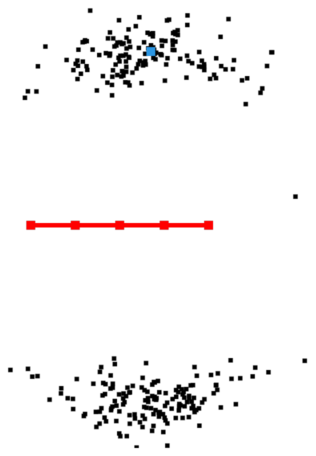
(a) At start no information is known and the particle set is empty.



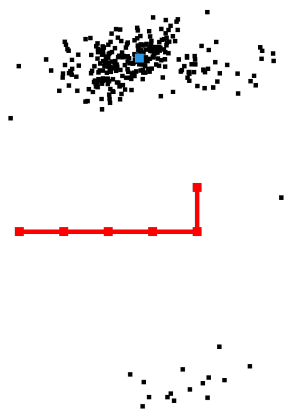
(b) After the user has taken a step, using the last range measurement, a particle cloud is initialised.



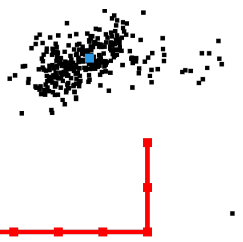
(c) On each step, new range measurements are incorporated in the filter by updating particle weights.



(d) As RSSI measurements do not contain angle information, multiple solutions can arise when walking in straight lines. This can result in mirroring problems.



(e) After a user makes a turn, the particle filter contains enough information to discard one of the solutions.



(f) Given enough movement, the filter eventually converges to a good estimate of the landmark location.

Figure 3.2.2: Initialisation of landmarks is done using a separate particle filter. Given range measurements and movement of a user the initial position of a landmark is estimated. Note that in these visualisations user positions are assumed to be known.

Given a measurement z we can update the EKF as follows. For each particle m we start with defining the transition between our state and our expected observation: this is the Euclidean distance between our estimate of the user and our estimate of the landmark:

$$h_m(x_b^{[m]}, y_b^{[m]}) = \sqrt{(x_{u,t}^{[m]} - x_b^{[m]})^2 + (y_{t,u}^{[m]} - y_b^{[m]})^2} \tag{3.2.8}$$

In other words, for any beacon b, the function $h_m(x_b^{[m]}, y_b^{[m]})$ defines the distance between that beacon and the user estimate $[x_{u,t}^{[m]}, y_{u,t}^{[m]}]^T$ of particle m where u resembles the user and t the current time step. As the user estimate is contant, given a particular particle, only the beacon's location is a variable in this function.

We then calculate the *innovation* which resembles the error from our state to the observation:

$$v = z - h_m \tag{3.2.9}$$

As the transition from our state to observation is non-linear we have to linearise. We define the Jacobian by computing the partial derivates of our transition function h_m in both x and y:

$$H = \frac{\partial h_m}{\partial [x_b^{[m]} y_b^{[m]}]} = [\frac{x_{u,t}^{[m]} - x_b^{[m]}}{h_m}, \frac{y_{u,t}^{[m]} - y_b^{[m]}}{h_m}]^T \tag{3.2.10}$$

For completeness, the full derivation of h_m in x is as follows (the derivation in y is equivalent):

$$h_m = \sqrt{u} \text{ where } u = dx^2 + dy^2 \tag{3.2.11}$$

$$\begin{aligned} \frac{\partial h_m}{\partial dx} &= \frac{1}{2\sqrt{u}} \frac{\partial u}{\partial dx} \\ &= \frac{dx}{\sqrt{dx^2 + dy^2}} \\ &= \frac{x_{u,t}^{[m]} - x_b^{[m]}}{h_m} \end{aligned} \tag{3.2.12}$$

Based on the Jacobian we calculate the Kalman gain. Note that, due to our static motion model for the landmarks, the covariance matrix estimate is not updated (i.e. $\bar{\Sigma}_{b,t}^{[m]} = \Sigma_{b,t}^{[m]}$).

$$\sigma_v = H \Sigma_{b,t}^{[m]} H^T + Q_t \tag{3.2.13}$$

$$K = \Sigma_{b,t}^{[m]} H^T \sigma_v^{-1} \tag{3.2.14}$$

Given the innovation and the Kalman gain we can update our state and covariance. As our transition model assumes static landmarks we directly update the previous state. The Kalman gain is used as a weighting mechanism.

$$[x_{t+1}^{[m]}, y_{t+1}^{[m]}]^T = [x_t^{[m]}, y_t^{[m]}]^T + Kv \tag{3.2.15}$$

$$\Sigma_{t+1}^{[m]} = \Sigma_t^{[m]} - K\sigma_v K^T \tag{3.2.16}$$

The last step is updating the importance weight of the particle:

$$w_t^{[m]} = w_{t-1}^{[m]} f(z|h_m, \sigma_z) \tag{3.2.17}$$

Updating the previous state and computing the weight completes processing the measurement. This process is performed for every landmark and for every particle.

3.2.4 Resampling

After each observation is processed all particles have been updated and contain new importance weights. We can now perform the resampling. However, it does not make sense to resample after each step; there is just to little information for the resample process. In order to overcome this we utilise *Sequential Importance Resampling (SIR)*. After we updated the importance weights and normalised them, we calculate the effective number of particles ($\hat{N}_{\text{eff}}$). If the effective number of particles drops below a given threshold we resample.

$$\hat{N}_{\text{eff}} = \frac{1}{\sum_{i=1}^{N}\left(w_k^{[i]}\right)^2} \tag{3.2.18}$$

When the filter needs to resample we make use of *low variance sampling* to sample new particles. This type of sampler only requires a single random number to make the selection[4]. The most important advantage of the low variance sampler is its time complexity: $O(M)$ where M is the amount of particles that needs to be selected. In comparison, roulette wheel selection has a complexity of $O(MlogM)$[5]. The algorithm for the low variance sampler can be found in Algorithm 3.3.

3.3 Complexity

One of the main requirements of the SLAC algorithm is that it should run online and in real time. To assess whether this is achievable by the current algorithm we look at the time complexity of the different components:

Pose sampling
: Pose sampling is a fixed process in which we only process incoming motion measurements; this results in a complexity of $O(1)$.

Initialisation filter
: The initialisation filter uses M_i particles to update a landmark. Updating a single particle has complexity $O(1)$. The complexity of the whole filter, for each individual landmark, is $O(M_i)$.

[4]In comparison to, for example, roulette wheel selection which chooses a set of random numbers equal to the amount of particles that needs to be selected.

[5]M random numbers have to be drawn and for each random number a particle has to be selected from the list.

Algorithm 3.3 Low variance sampling. Input consists of the desired number of particles N, the importance weights W of the current particle set P.

```
 1: function LowVarianceSampler($N, W, P$)
 2:     $M = |W|$
 3:     $W = \text{normalise}(W)$
 4:     $r = \frac{random()}{M}$        ▷ $random()$ returns a uniform random number between 0 and 1
 5:     $c = W_0$
 6:     $i = 0$
 7:     $S = \emptyset$

 8:     for $m = 1$ to $N$ do
 9:         $u = r + \frac{m-1}{M}$

10:         while $u > c$ do
11:             $i = i + 1$
12:             $c = c + W_i$
13:         end while

14:         $S = S \cup P_i$
15:     end for

16:     return $S$
17: end function
```

EKF update
Each individual two-dimensional EKF is updated using a fixed set of computations which results in a complexity of $O(1)$.

Low variance sampling
As we described in the previous section, the complexity of the sampler is equal to $O(M_l)$ where M_l is the number of particles used to represent the user and the landmarks.

The SLAC algorithm, FastSLAM alike, keeps track of $1 + MN$ filters where M is the number of particles and N the number of landmarks that has already been initialised. This bring us to a total complexity of $O(MN + M_i)$. At start M will be zero (no initialised particles) but will increase when more landmarks are initialised. Subsequently, M_i will eventually drop to zero if an estimate of all landmarks has been found. The time complexity of $O(MN + M_i)$ does not include resampling, but this process does not take place on every step and has therefore a smaller impact.

3.4 Moving landmarks

In real world scenario's, landmarks (i.e. devices) will, on occasion and some more than others, move. When a device is slowly moved and is able to continuously broadcast messages the EKF will slowly adapt to the change in position. While there will be a expected drop in accuracy the update step of the EKF will eventually find the new position.

Unfortunately, devices will generally not move slowly but will abruptly disappear and appear at a different location; this is the case when a device is plugged out, moved and then plugged in again. In such a scenario the RSSI signal will be drastically different.

To adapt to these situations we utilise a simple technique: Each landmark signals that something has changed when it starts up again after a power loss. The SLAC algorithm detects this message and then reinitialises that particular landmark.

Using the factorised approach, when a landmark is moved, only that single landmark has to be reinitialised by removing the EKF and restarting the initialisation filter. All the information of the other landmarks can be retained. This makes adapting to moving landmarks efficient.

Of course, by acting on a message sent by the device a part of the responsibility moves from the algorithm to the devices we want to locate. We opted for this choice to highlight the efficiency and the way the algorithm can react to changes. Detection of a moving device can also be done as part of the algorithm; for example by evaluating the change in RSSI signal. These kind of measures are however outside the scope of this research.

Chapter 4

Implementation details & evaluation

In this chapter we explain details regarding the actual implementation of SLAC. Whereas in the previous chapter we focussed on how the system works, here we constrain us to the specific implementation used in this project. Second, we explain how this implementation has been evaluated using both simulations and real world experiments.

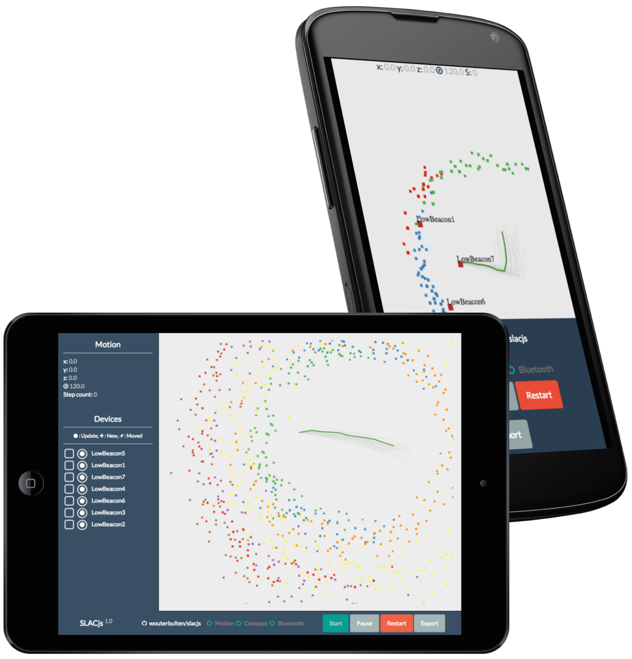


Figure 4.0.1: Screenshots of the SLAC application running on both an *iOS* tablet and an *Android* mobile phone.

4.1 Javascript implementation

SLAC has been fully implemented in *Javascript* and more specifically using the *ECMAScript version 6/2015* standard. Javascript has been chosen to support a large range of devices on which the algorithm can run; this includes web browsers and mobile phones. The *Apache Cordova* platform[1] was used to access native API's of mobile devices such as the Bluetooth radio and the motion sensors. This link between native components and the Javascript implementation is show in Figure 4.1.1.

The implementation of the SLAC algorithm is an event-based system where the motion data controls the flow. Every time the pedometer detects a new step a single run of the algorithm is performed. All the RSSI measurements that were recorded between the last and the current step are used in the update.

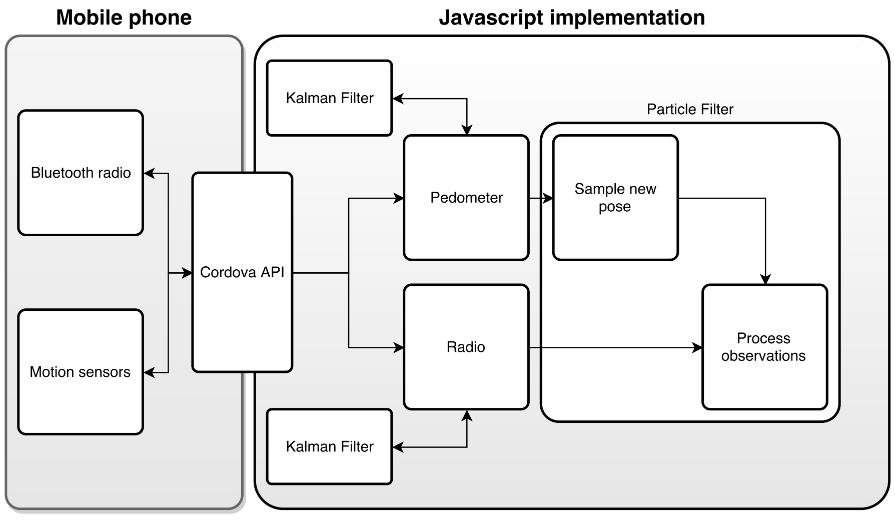


Figure 4.1.1: Visualisation of the Javascript implementation. The Cordova API links the native bluetooth and motion modules (left) to the Javascript components (right). The data flow is left to right: sensor data flows from the phone, through the API to the particle filter.

Figure 4.0.1 shows an example of the application running on both a tablet and a mobile phone. Figure 4.1.2 shows an annotated screenshot of the application's interface. The full source code is available online[2] and is licensed under the *GNU Lesser General Public License* so that it can be used and extended in other projects.

To show SLAC in action a recording was made from one of the live runs. In this run a user walks around inside a room where seven devices were placed in wall sockets. During the run two devices are disconnected and placed into another wall socket; the video shows how the algorithm reacts to this change by reinitialising those devices. The video can be found online at:

`https://www.youtube.com/watch?v=0Oq2QNOFz8U`

Additionally to the video, a live demonstration of SLAC running in the browser (using simulated data) can be found online:

`https://wouterbulten.nl/slacjs/`

[1] `https://cordova.apache.org/`
[2] `https://github.com/wouterbulten/slacjs`

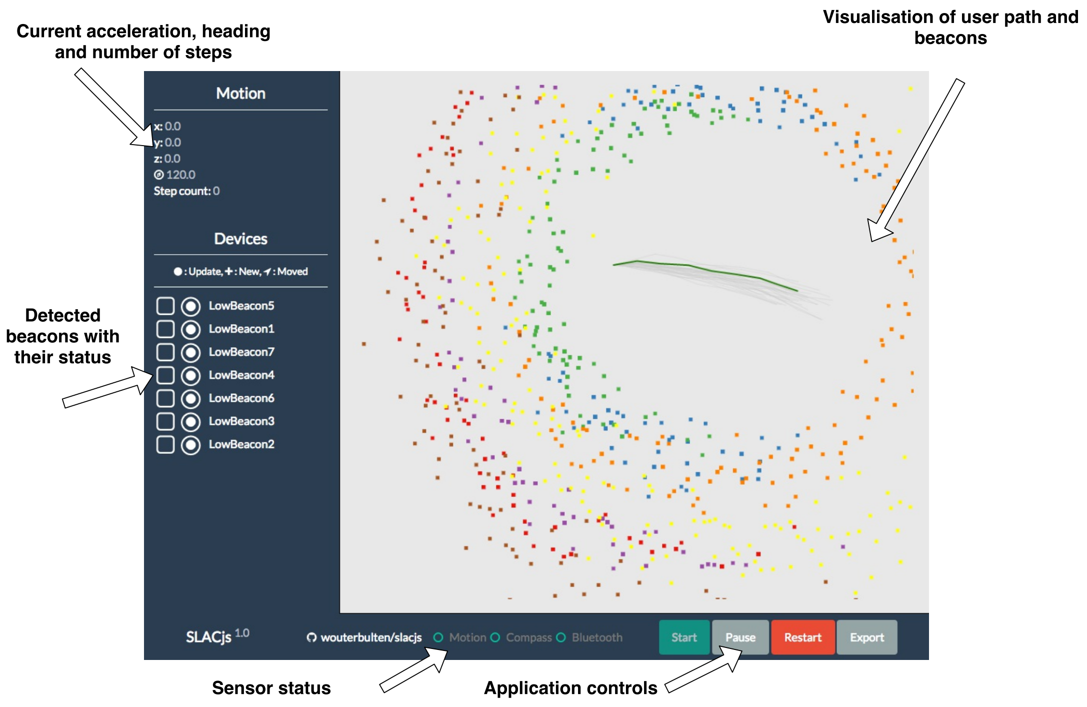


Figure 4.1.2: Screenshot of the SLAC application running on a tablet (with annotations). The sidebar shows the current motion data (acceleration, heading and step count) and an overview of each device. Icons on the device overview indicate whether a device has been moved, is new or sent out a new RSSI update. The right part of the application shows the current state. The green line shows the best estimated path of the user. All coloured squares show initialisation filters for the devices (each color represents a single device).

4.2 Simulations

The SLAC system was first evaluated using simulations; this granted the opportunity to repeat the experiment and control the environment. In the simulation we emulated the world by building an environment of similar dimensions as one of our real world test beds. On the same coordinates as in the real world simulated landmarks were placed. See section 4.3 for a description of this environment. Each landmark broadcasts messages with a signal strength following the path loss model (with added noise).

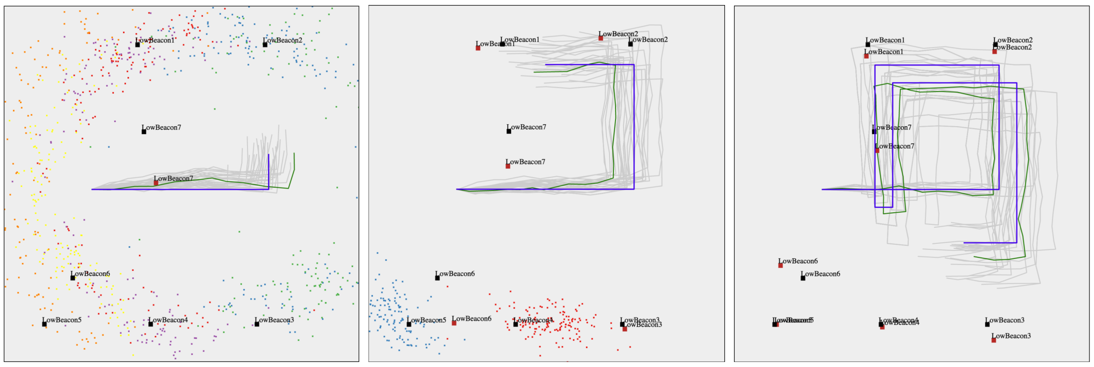
In this simulated environment we let a user walk a fixed path and run the SLAC algorithm on every step. The path is a double loop (with the second part reversed) and is shown in Figure 4.2.1. We use a fixed step size of $0.5m$ and always perform 66 steps[3].

For the user motion we introduce two scenarios. In the first scenario the algorithm uses the simulated user movement as a direct input; this resembles a world were we have perfect motion information. The second scenario adds noise[4] to the user movement before this is used as an input.

The second input, RSSI, is also varied. For the RSSI measurements there are a total of seven conditions: we vary the number of RSSI updates each landmark broad-

[3]This number and pattern was chosen to resemble the path that is walked during the live tests.

[4]Gaussian noise on both the step size and the heading.



(a) Beginning of the simulation, only a single landmark has been initialised.

(b) After more movement and measurements all but two landmarks have been found.

(c) Further refining of landmark estimates is done using separate EKFs.

Figure 4.2.1: Three screenshots of a SLAC simulation at various stages. The blue line signals the ground truth of the human movement. Each grey line represents a particle's estimate of the user path. The green line is the current best estimate. Black squares are the ground truth of landmark locations, red blocks represent the current best estimate. Coloured small squares at screen one and two represent individual initialisation filters.

casts between each consecutive algorithm step. As the signal strength is used to make range estimates the number of received messages could have an effect on the overall performance. The different settings are: $1, 2, 5, 10, 25, 50$ and 100 updates per step.

In total 14 specific (2 motion $\times$ 7 RSSI) settings are tested. Each setting is simulated 500 times.

4.3 Online recordings, offline evaluation

Simulating RSSI values and movement has its drawbacks: noise is predictable and there is less interference from events in the environment. In general it is hard to fully simulate all the factors of a real world environment. In order to asses the performance of the algorithm outside a simulated world, three different locations were used to test the algorithm in the wild.

While SLAC runs online and in real time the data at these three locations has been recorded and analysed offline at a later stage. The algorithm did however run during the data collection to give feedback about the process.

Each recorded data set consists of the raw unprocessed and unfiltered motion data (i.e. acceleration and heading) and RSSI measurements. Each data point has a timestamp which is used to play back that particular measurement at the correct time. These datasets are played back several times to get an average performance. This is particular important as the algorithm is a random process: using the same input data twice will result in different outcomes.

Not all data that has been recorded is used for the evaluation. First, we opted to only use datasets that have user paths that walk past or close to all landmarks. Especially in large environments not all landmarks are visible all the time. To be able to correctly compare environments we only look at runs where each landmark could have been found[5]. Second, due to a problem with the data recording some runs also contained

[5]To stress: We selected datasets were landmarks are visible during the run (e.g. at least a single RSSI measurement). This does not mean that the algorithm always finds an estimate of its location.

Table 4.3.1: Description of all runs carried out at the *Almende* office. The step count gives a rough indication of the length of each run.

Run ID	Description	Step count
A_1	Double loop in the center of the room with the second reversed	53
A_2	Double loop in the center of the room with the second reversed	65
A_3	Double loop in the center of the room with the second reversed	61
A_4	Double loop in the center of the room with the second reversed	64

motion data from previous runs; these datasets have also been left out. Datasets were *not* discarded for accuracy reasons.

4.3.1 Live tests at Almende

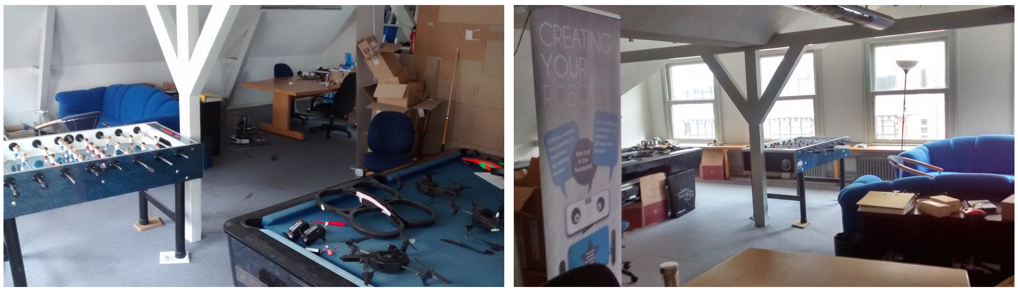


Figure 4.3.1: The top floor at the Almende building was used as one of the testbeds. Seven landmarks were placed in wall sockets around the room. No furniture or objects were removed before testing. The room's dimensions are around $10 \times 10m$.

The first live tests are done at the building of Almende[6]; see Figure 4.3.1 for a picture of the environment. Seven beacons were placed in the environment; five of them in wall sockets and two using extension cords. A user walked around in this environment following a fixed pattern: looping around the foosball table and back. This path resembles that path used in the simulations. See Figure 4.2.1 for an overview of the landmark locations and Table 4.3.1 for an overview of the specific runs.

4.3.2 Rotterdam Community Solutions

A second live test was performed at the company *Rotterdam CS*[7] who granted us access to their office. Fourteen beacons were placed around the office floor (a semi-open space with dimensions around $23 \times 17m$). A 3D rendering of the location is displayed in Figure 4.3.2. A description of all the runs that have been recorded are described in Table 4.3.2.

During testing the office was in use and employees walked around. All the landmarks were placed in wall sockets with two of them in the middle of the room and the remaining twelve in the outer walls. Most landmarks were placed around work stations of employees.

[6]`http://www.almende.org`
[7]`https://www.rotterdam-cs.com`

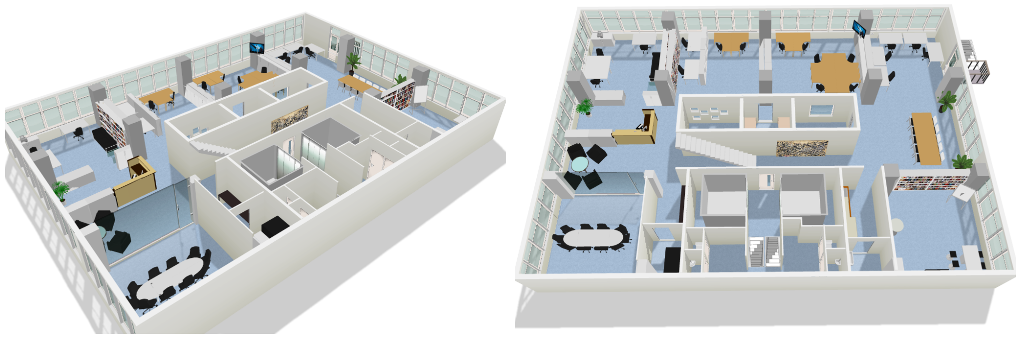


Figure 4.3.2: The full office of Rotterdam CS was used as the second testbed.

Table 4.3.2: Description of all runs carried out at the *Rotterdam CS* office. The step count gives a rough indication of the length of each run.

Run ID	Description	Step count
CS_1	Single loop around the center of the office	43
CS_2	Double loop around the center of the office	82
CS_3	Three loops around the center of the office	119
CS_4	Double loop around the center of the office	75
CS_5	Double loop around the center of the office	75
CS_6	Random walk around all nodes	188
CS_7	Three loops around the center of the office	140
CS_8	Double loop, with the second reversed	99

4.3.3 Radboud University

The third and last location of our live tests was the basement of the Faculty of Social Sciences at the *Radboud University*. These tests were carried out on a Saturday which resulted in an almost empty building. Fourteen landmarks were placed in wall sockets in various corridors of the basement; the total space was around $50 \times 30m$.

The whole basement consists of many small corridors and a lack of (large) open rooms. This location is therefore mainly used to measure the effect of walls on the accuracy of the localisation. See Figure 4.3.3 for an example and Figure 4.3.4 for an overview of the landmark locations.

Table 4.3.3: Description of all runs carried out at the *Spinoza* building. The step count gives a rough indication of the length of each run.

Run ID	Description	Step count
SP_1	Full loop through the environment.	419
SP_2	Full loop. More natural movement (i.e. less straight paths and turns).	372
SP_3	Several loops through the full space	1014

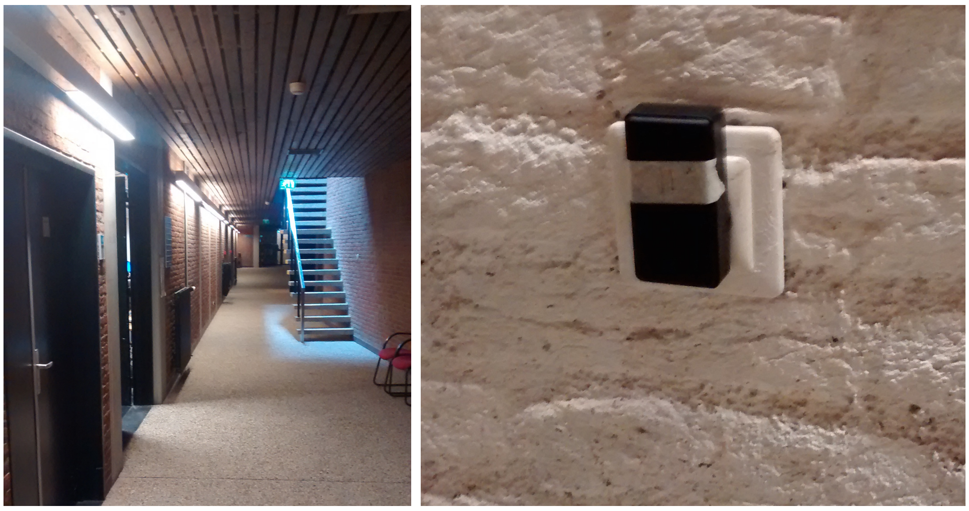


Figure 4.3.3: The basement of the Faculty of Social Sciences at the Radboud University was used as a test for larger environments. In total, fourteen landmarks were placed in wall sockets (right picture.

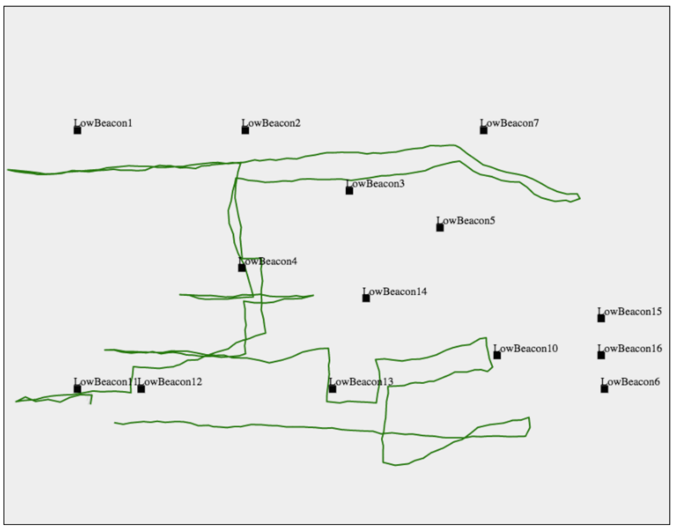


Figure 4.3.4: Ground truth locations of landmarks (black squares) in the Spinoza building. The green line is an example of the estimated path of the user from one of the test runs.

Chapter 5

Results

This chapter is separated in two parts: first we describe the results from our simulations and the specific conditions that have been tested. Second, we show the results from the three live tests that have been carried out.

5.1 Simulation results

Fourteen distinct simulations have been run where two factors were varied: the movement noise and the number of RSSI updates per algorithm step. These two factors, and their effect, are described in the next two sections. Overall, the best result was obtained in the simulation without movement noise and with 100 RSSI updates per step. This resulted in an average localisation error of $.16m$; i.e. the estimation of landmark positions differend, on average, only 16 centimetres from the ground truth. The same condition, but with movement noise, resulted in an average error of $.46m$. Table 5.5.1 contains a full overview of all the simulation runs.

5.1.1 Effect of movement noise

An ANCOVA was conducted to determine a statistically significant difference between simulations with and without movement noise on the average localisation error controlling for the number of RSSI updates per step. As the average localisation error is not normally distributed for some of the conditions (see Figure 5.1.1) each value has been converted using a log_{10} transition.

There is a significant effect of simulation type on the localisation error after controlling for the number of RSSI updates, $F(1, 6997) = 66.804, p < .000$ but the effect size is small ($eta^2 = 0.065$). The average localisation errors for simulations respectively with and without noise are 1.6033 and 1.4503 meters. When correcting for the number of RSSI updates, simulations with movement noise have on average higher localisation errors. The effect of number of RSSI updates has a large significant effect ($t(6997) = 6092.416, p < .000, eta^2 = .465$).

See Figure 5.1.2 for an overview.

5.1.2 Effect of number of RSSI updates

An inverse curve estimation model was used to predict the average localisation error, given noisy movement measurements, based on the number of RSSI updates per step.

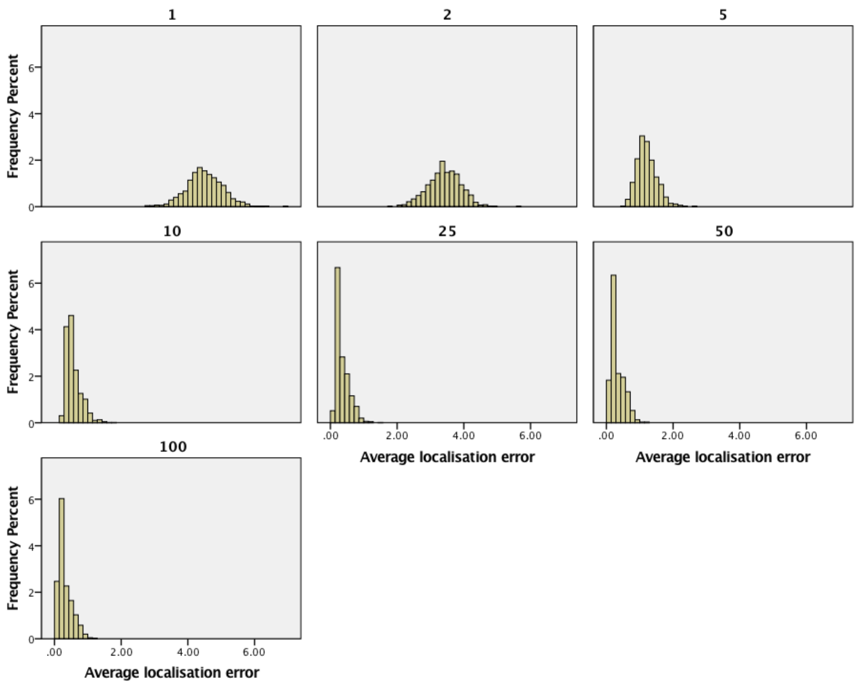


Figure 5.1.1: Localisation error is not normally distributed for some of the conditions. For high update rates the localisation error cannot improve anymore which results in a skewed distribution.

A significant regression equation was found ($F(1, 3498) = 33868.750, p < .000$), with an R^2 of .906. The average localisation error is equal to $.431 + \frac{4.390}{\text{RSSI}}$ meters with RSSI the number of updates per step.

Likewise, for the perfect movement condition, a second curve estimation model was used to predict the average localisation error. A significant regression equation was found ($F(1, 3498) = 39472.204, p < .000$), with an R^2 of .919. The average localisation error is equal to $.175 + \frac{4.772}{\text{RSSI}}$ meters with RSSI the number of updates per step.

5.1.3 Difference between landmark locations

In the simulations the only difference between landmarks is their location which has an effect on how close the user is and how often it passes. For each condition (RSSI updates per step) a repeated-measure-ANOVA was conducted for the simulations with noisy movement on the average localisation error using landmark position as the within-subject factor. As the individual localisation errors are not normally distributed for some of the conditions (see Figure 5.1.1), each value has been converted using a log_{10} transition.

Mauchly's Test of Sphericity has been used to detect violations of the sphericity assumption and the Greenhouse-Geisser correction was applied. See Table 5.1.1 for an overview.

For each condition there was a significant effect of landmark position on the average localisation error, see Table 5.1.2. The size of the effect ranged from very large ($eta^2 > .2$) for conditions with only few RSSI updates to small ($eta^2 < .1$) for conditions with many RSSI updates. In other words, there is a significant effect of landmark position but this effect declines with an increasing number of RSSI updates per step.

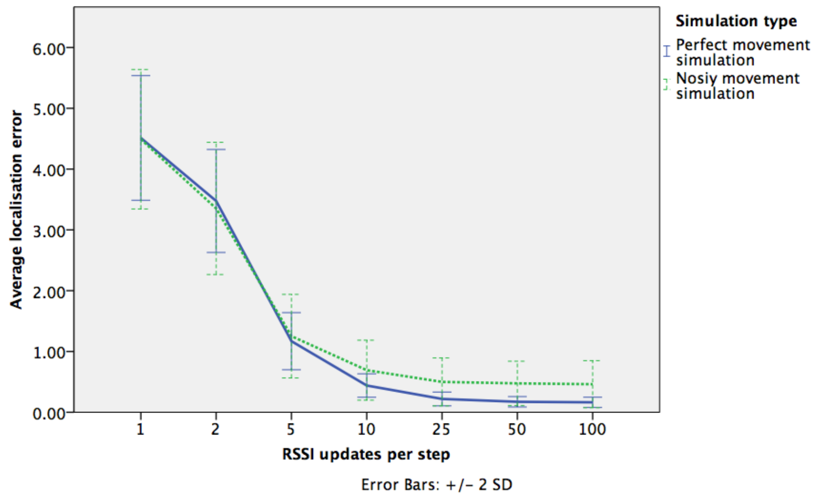


Figure 5.1.2: Localisation error (averaged over all landmarks) for simulations with different RSSI frequencies. The dashed line (top) represents simulations with noisy movement information; the other has perfect motion data. The decline in localisation error shows the intuitive result that more data improves the localisation.

Table 5.1.1: Results of Mauchly's Test of Sphericity and the Greenhouse-Geisser correction that is applied for each condition for the repeated-measures-ANOVA.

RSSI updates	Mauchly's Test of Sphericity	Greenhouse-Geisser correction
1	$\chi^2(20) = 803.424, p = .000$	.597
2	$\chi^2(20) = 483.307, p = .000$	.746
5	$\chi^2(20) = 154.974, p = .000$	.913
10	$\chi^2(20) = 117.418, p = .000$	.921
25	$\chi^2(20) = 242.841, p = .000$	.843
50	$\chi^2(20) = 245.568, p = .000$	.848
100	$\chi^2(20) = 208.164, p = .000$	.862

5.2 Live test: Almende

The four live tests at Almende resulted in 2000 data points (500 per test). Of these 2000 runs 108 did not fully initialise all beacons. Based on the step count and the number of RSSI updates the average number of RSSI updates per step lays between 7 and 10. The results of the Almende runs will therefore be compared with the *5* and *10 RSSI updates* condition of the noisy simulation. An overview of the individual runs in comparison to the simulations is shown in Figure 5.2.1.

An independent-samples t-test was conducted to compare the average localisation error in the live tests with the *5 RSSI updates* simulations. There was a significant difference ($t(2498) = -59.527, p = .000$) in average localisation error for the live runs ($M = 2.29, sd = .44$) and simulations ($M = 1.25, sd = .34$).

A second independent-samples t-test was conducted to compare the average localisation error in the live tests with the *10 RSSI updates* simulations. There was a significant difference ($t(2498) = -78.524, p = .000$) in average localisation error for the live runs ($M = 2.29, sd = .44$) and simulations ($M = .69, sd = .25$).

Combined we can state that there is a significant difference between simulations and

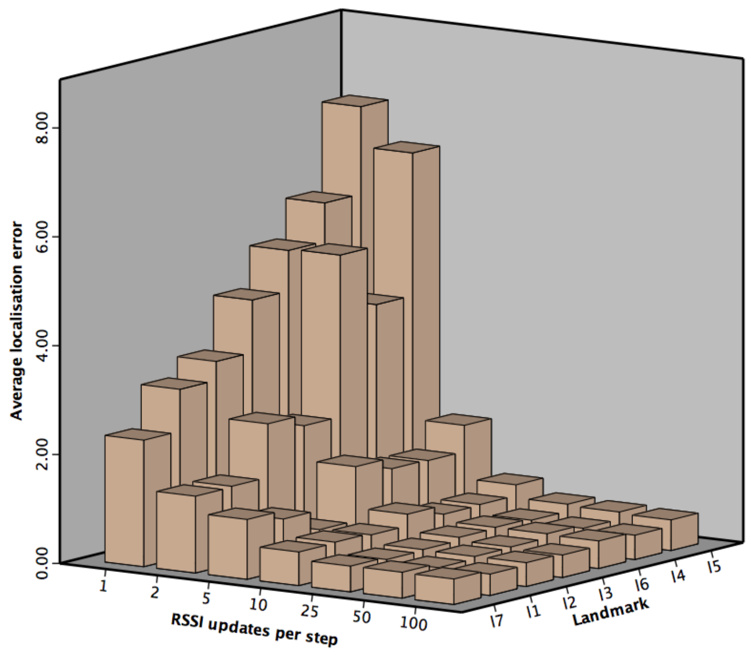


Figure 5.1.3: Localisation error (separated per landmark) for simulations with different RSSI frequencies. Only the data from the noisy movement simulations has been used for this figure.

real life tests ranging between 1.04 and 1.6m.

5.3 Live test: Rotterdam CS

None of the runs of the Rotterdam CS tests managed to initialise all beacons. Averaged over all runs, 53.4% of beacons were not initialised but this is highly dependent on the specific run.

We do have to note that we evaluated all landmarks even though not all of them are close to the paths of the users in the runs. In Figure 5.3.2 the pattern of the missing data is shown together with a bar plot of the frequency that a specific landmark was not initialised. The differences are clear: some landmarks are almost never initialised where

Table 5.1.2: Results of each repeated-measure-ANOVA conducted on the simulations with noisy movement using landmark position as the within-subject factor.

RSSI updates	**Effect**		**Partial** eta^2
1	$F(3.584, 1369.246) = 386.167$	$p = .000$	.503
2	$F(4.473, 2227.97) = 1152.524$	$p = .000$	.698
5	$F(5.475, 2227.97) = 198.102$	$p = .000$	.284
10	$F(5.529, 2758.900) = 33.274$	$p = .000$	.063
25	$F(5.060, 2525.068) = 27.325$	$p = .000$	.052
50	$F(5.087, 2538.630) = 34.545$	$p = .000$	.065
100	$F(5.170, 2580.006) = 34.544$	$p = .000$	.065

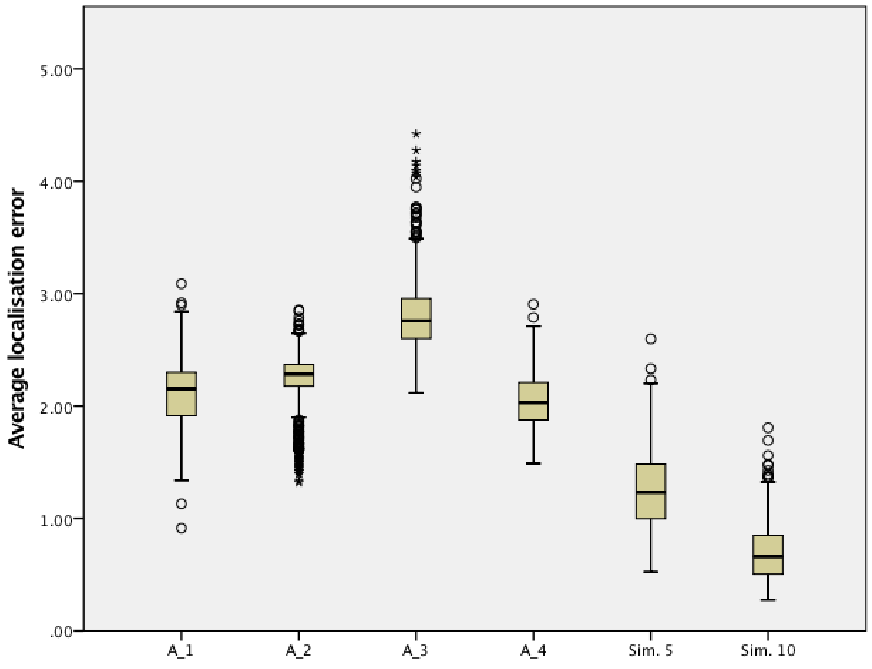


Figure 5.2.1: Showing the average localisation error of the four Almende runs (first four) in comparison to the simulations (last two).

for others an estimate is almost always found.

In Figure 5.3.2 the performance of each individual run in shown. As the number of initialised landmarks differs from run to run the results should be viewed given that fact in mind.

5.4 Live test Spinoza

Out of the 1500 cases of the Spinoza test only 1 did not manage to fully initialise all landmarks. The performance of each individual run in shown in Figure 5.4.1.

The run using more natural walking patterns ("SP_2") has the lowest accuracy. It is however not clear whether this is caused by the walking pattern or other influences. Moreover, this run has also the lowest step count (and with that fewer iterations of the algorithm) which could also account for the lower accuracy.

5.5 Overview of results

Table 5.5.1 shows an overview of the localisation errors (i.e. the performance) of the simulations in comparison to the live tests carried out in the Almende building. Table 5.5.2 shows the results of all the tests at the Rotterdam CS building. In these tests a large part part of the landmarks did not initialise, and therefore did not have a position estimate, this table also shows the percentage of landmarks that did have an estimate. The last table, Table 5.5.3, shows the results of all the runs done at the Spinoza building.

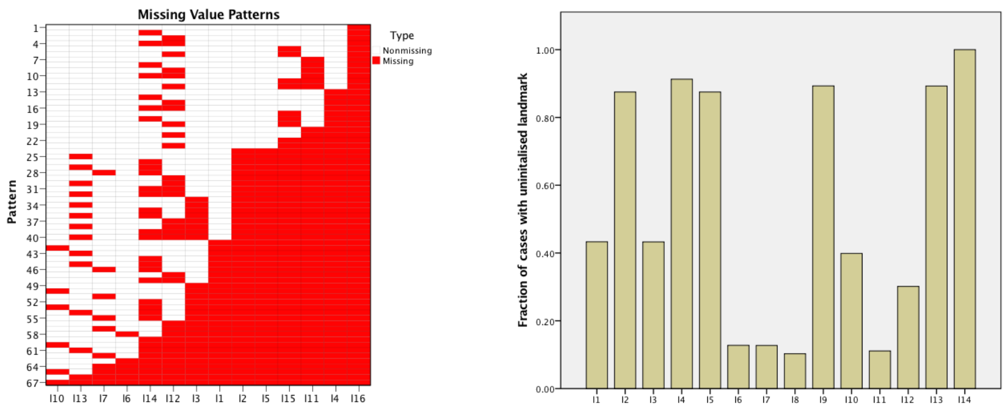


Figure 5.3.1: Showing missing data in the Rotterdam CS dataset. The left figure shows specific data patterns, each filled square indicates that an estimate of that landmark is missing. The bar plot on the right shows the percentage of time a specific landmark was not initialised.

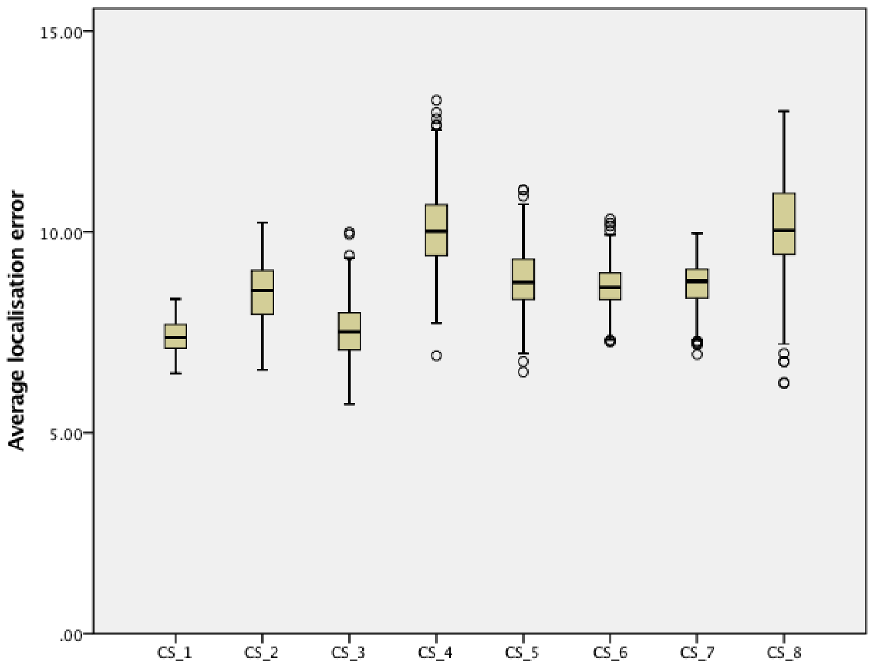


Figure 5.3.2: Showing the average localisation error of the Rotterdam CS runs. Note that these localisation errors are only based on the initialised landmarks. In the Rotterdam CS tests many of the landmarks did not initialise before the end of the test.

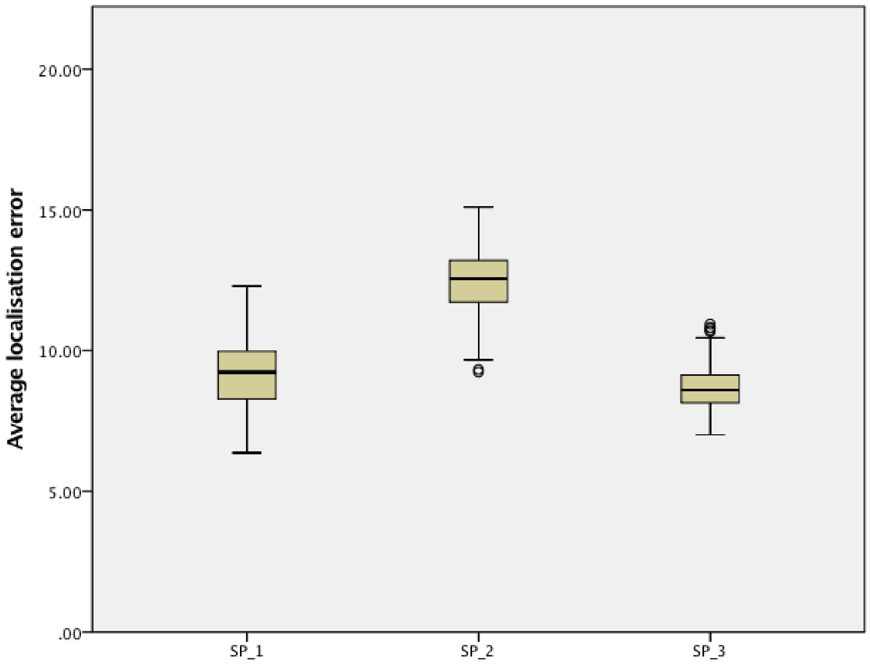


Figure 5.4.1: Showing the average localisation error of the Spinoza runs. The error is based on all landmarks as these were in 99.93% of the cases all initialised.

Table 5.5.1: Overview of the simulation runs and the corresponding live tests (at the Almende building). For each run the average localisation error is reported. The 'complete' column of the table describes the percentage of runs that succeeded in initialising all landmarks (i.e. completing the initialisation part and continuing with the EKF).

Run	**Average localisation error** (m)	**Standard deviation of error** (m)	**Complete** (%)
Sim. 1 RSSI update	4.51	.51	.86
Sim. 2 RSSI updates	3.48	.42	100
Sim. 5 RSSI updates	1.17	.23	100
Sim. 10 RSSI updates	.44	.10	100
Sim. 25 RSSI updates	.22	.06	100
Sim. 50 RSSI updates	.17	.04	100
Sim. 100 RSSI updates	.16	.04	100
Sim. with noise, 1 RSSI update	4.49	.57	75
Sim. with noise, 2 RSSI updates	3.35	.54	99
Sim. with noise, 5 RSSI updates	1.25	.34	100
Sim. with noise, 10 RSSI updates	.69	.25	100
Sim. with noise, 25 RSSI updates	.50	.20	100
Sim. with noise, 50 RSSI updates	.48	.18	100
Sim. with noise, 100 RSSI updates	.46	.19	100
Almende 1	2.10	.31	95
Almende 2	2.20	.30	100
Almende 3	2.84	.36	92
Almende 4	2.04	.25	100

Table 5.5.2: Overview of all the runs at the Rotterdam CS building. For each run the average localisation error is reported. At these live tests many of the landmarks did not initialise (e.g. an estimate was not yet found within the time frame of the test). Therefore the average amount of beacons that was initialised (averaged over all runs for that particular test) is reported in the last column.

Run	**Average localisation error** (m)	**Standard deviation of error** (m)	**Beacons initialised** (%)
Rotterdam CS 1	7.41	.39	3
Rotterdam CS 2	8.50	.69	35
Rotterdam CS 3	7.54	.68	54
Rotterdam CS 4	10.11	.95	40
Rotterdam CS 5	8.81	.76	48
Rotterdam CS 6	8.64	.52	85
Rotterdam CS 7	8.69	.50	55
Rotterdam CS 8	10.14	1.19	53

Table 5.5.3: Overview of all the runs at the Spinoza building. For each run the average localisation error is reported. The last column shows the percentage of runs that succeeded in finding an estimate for all beacons.

Run	**Average localisation error** (m)	**Standard deviation of error** (m)	**Complete** (%)
Spinoza 1	9.14	1.09	100
Spinoza 2	12.48	1.02	100
Spinoza 3	8.64	.72	100

Chapter 6

Discussion

In this thesis we set out to explore the possibility of designing a fully autonomous self-localisation system for a network of devices by utilising the users of the space that the network is deployed in. Our main goal was to perform this localisation without any prior information about the network.

We have shown, through simulations, that within controlled environments (i.e. Gaussian noise, fixed paths of users, no obstacles), we can achieve an average localisation error below .20m. This means that, after running the algorithm, the estimate of a device's location will, on average, only be 20 centimeters away from its actual position. The only input that is used are signal strength measurements from the devices and motion data from a user. No information about the structure of the room or the position of the device was used.

When movement noise was introduced in these simulations, i.e. there is no ground truth movement information available, the performance drops slightly but is still accurate: on average a localisation error below .5m is achieved.

These simulations show the maximum attainable performance; they were conducted in controlled environments using Gaussian noise and a high update frequency of the devices. When the update frequency of devices is lowered, to a level similar to our live tests, the average localisation error increased and resulted in an average error of .69 to 1.25m.

After simulations the algorithm was live tested in a real world environment with the same number of devices and of equal dimensions. Our test environment was, however, not a clear open space as our simulation environment was but contained many sources of noise, including objects, walls and people walking around. This was deliberate as these noisy environments are exactly the target platform for this technique. All combined, our live tests showed a localisation error of 2.3m, averaged over all devices. This result is good enough for room level accuracy, but there is room for improvement. These results where achieved by letting a single user walk around for one to two minutes (roughly 60 steps).

Additionally two other locations were used as test beds. These locations were larger, contained other users and consisted of multiple rooms. The localisation errors of devices in these environments were larger than in our other test (with an average localisation error over 8m). Also, more devices did not finish the initialisation phase before the end of the test run. These errors are too large to achieve room level accuracy which suggests that for larger spaces we need to take additional measures.

6.1 Evaluation of requirements

Besides performance, we started this research with three qualitative measures or requirements: the localisation should be online, privacy must be assured and no prior information should be used.

The SLAC system is a fully online method. SLAC runs online and gives instantaneous estimates of device and users positions in contrast to offline methods which only give these estimates after processing all data. This characteristic is the result of the SLAM approach and the use of Rao-Blackwellized filters; only the last state is required to predict a new state.

All computations are local and sharing is possible. The whole algorithm runs locally on a user's device without communicating with the devices it tries to locate. The only input, from sources outside the user's own device, are RSSI measurements. These measurements are one-way communications.

We have to note that by using a wireless device inside a building a user can still be tracked by external measures. This is however even the case without running the SLAC algorithm. It is key that for the localisation itself no external communication is required. Moreover, the current setup makes it easy to share device estimations without sharing a user's path (we will go deeper in to this subject in section 6.4).

No anchors, additional hardware or other information is required. The only information the SLAC algorithm needs is an estimate of the step size of the user (although a general average could be used for this) and the signal strength of devices at a 1 meter distance. This last value is often part of the wireless protocol and part of the broadcast message. In other words, the localisation can take place without prior information.

6.2 Factors influencing performance

The localisation error depends on many factors, of which some could be tested in our simulated environment. We also found (large) differences between different test beds and individual runs. The most intuitive factor is probably the length of our tests. Especially in the large environments many devices were not initialised due to a shortage in measurements. The following factors can explain these differences:

More RSSI updates increases performances. We found, using controlled simulations, that the number of RSSI updates per algorithm step has a very high effect on the performance of the system. This follows directly from our system: given more information the Kalman filter responsible for filtering the raw RSSI signal will be able to give a less noisy estimate of the current distance to a beacon. These distance estimates are vital in updating landmark positions and weighing particles.

While the effect of the number of RSSI updates is high, this effect eventually wears out. Given enough measurements the noise will eventually be filtered out and more measurements will not improve the distance estimate.

The two factors in the simulations, motion noise and number of RSSI updates, also show an interesting interaction: The movement noise only seems to influence performance when the number of RSSI updates is high. When the update frequency of devices is low the noise in the filtered distance measurements becomes so high that the movement noise does not add any noticable error.

The best performing simulations used high update frequencies (ranging from 25 to 100 updates per algorithm step). In our real world tests we only received, on average, seven to ten updates per step[1]. This suggests that increasing the update rate can improve performance in real world scenarios.

Devices' positions influence performance Within the simulations the only difference between devices were their positions and followed from that the relative path of the user. For the initialisation and updating the estimate it is important that a user passes a device from different angles. All other factors were controlled, e.g. the path of the user was constant, RSSI values followed the same function and there were no obstacles. Given this, we found significant differences between localisation errors of landmarks. Especially when the number of RSSI updates is low, the size of this effect is large.

These results indicate that we, in live tests, can expect differences in performance for individual devices and that the path of the user influences this.

Path loss model does not account for obstructions. In our simulated world Gaussian noise was added to RSSI values to simulate real world noise. All the live test environments contained objects, walls and humans who walked around. Each of these add a factor of unpredictable noise to the RSSI measurements.

Our live tests at the Almende building took place inside a single room; i.e. all devices were in the same room as the test. In our other test sites, for example the Spinoza building, the devices were spread out over a large floor with many walls between them. The path loss model that was used to translate RSSI to distance does not account for walls while, in real life, the actual RSSI value is influenced by them.

Motion modelling is inaccurate. In our simulations motion noise was modelled by adding noise to the ground truth. In the live tests this noise is, however, less nicely distributed and more dependent on environmental factors. Compass data for example, that is used to measure the current heading, can be influenced by nearby metal or magnetic objects. In robotics this mismatch between simulation and real world is also called the *reality gap* (Jakobi et al., 1995). Moreover, where the simulated user had a fixed step size, real users will vary their step size continuously; this adds an additional error to the estimate of the travelled distance. A large part of these errors are filtered out by the particle filter but it does not result in the very accurate modelling found in the simulations.

Short traces are enough for small dense environments; large environments need longer traces. While this result is intuitive it is still important to report: larger environments need more data for good estimates of devices locations. At our first live test a location estimate was found for all devices, in our second live test there was a large portion of devices whose estimate could not be found. The third test, which also was a larger environment but had longer traces, did however result in estimates for all devices. This indicates that for large environments, especially when device density is low, a longer trace is needed to get an initial estimate of a device's location.

[1]This was partially caused by limitations in our Javascript implementation.

The question how long these traces have to be remains. Unfortunately this is very dependent on the specific environment and, even more important, the location of the devices in this environment. To find an initial estimate of a device's position a user must pass a device from multiple angles (see Figure 3.2.2 for an explanation of this) and this depends on the path user takes; i.e. the time a device needs to initialise varies from device to device.

Noisy measurements result in mirroring errors. The distance estimates of devices is noisy, especially given obstacles present in the environment. However, a combination of the filters in the algorithm should be able to overcome this. There does, however, exist a situation where the estimated distance can be perfect but the localisation error very high: a mirrored position estimate. Such a mirroring error occurs when a device is initialised on a position mirrored to its actual position. We found that these errors are difficult to overcome within the small timeframes of our live tests.

6.3 Comparison to existing techniques

Most of the literature regarding the localisation of devices inside buildings focus solely on the devices themselves. Likewise, many of the algorithms focussing on users do not locate devices. It is therefore hard to make a fair comparison between SLAC and existing techniques. Nonetheless, we can give a broad overview of the localisation errors of the techniques we discussed previously in Section 2.

Focussing on users first, there are many methods to perform the localisation (see Section 2.2.2). WiFi-SLAM, using GP-LVM methods, by Ferris et al. (2007) achieve a localisation error, without labeled training data, of $3.97m$. The method using environment fingerprints as used by Yang et al. (2012) achieve a localisation error (for users) of $5.88m$. A offline SLAM approach, using GraphSLAM, by Huang et al. (2011) achieve a localisation error of $2.18m$. Most of these errors are averaged over the whole user path or calculated using specific points in the space[2].

The work of Sun et al. (2009) is, regarding implementation, closely related to the current research. Using robots their average localisation error for real world tests is around $1.3m$. This localisation error is however the error of the robot's path estimate and not of the landmark locations.

For locating devices most research focusses on using robots or on networks localising itself (see Section 2.2.1). Hollinger and Djugash (2008) and Hollinger et al. (2011), who use GP-LVM to simultaneously locate users and devices, achieve an average localisation error of around $4m$ for devices without using any odometry data (which makes the localisation harder). However, they do use ranging radios which results in better distance estimates. Using odometry data and EKF-SLAM a localisation error of around $3.3m$ was found.

One of the best results is achieved by Menegatti et al. (2010) who report an average localisation error in their experiments of $0.46m$; a camera and robot odometry is used to improve their results. A second good example is the work of Torres-González et al. (2014) who achieve a localisation error of around $0.2m$ using Sparse Extended Information

[2]An often used method to compute the accuracy of a trace is to see how the algorithm performs if the users is in a location for the second time. In a perfect system the two position estimates should be the same, the difference between the two estimates is often taken as the error.

Filters and robots. These very low errors are partially the result of incorporating inter-device measurements (i.e. distance estimates between devices).

Given our simulation and live test results, the SLAC algorithm sits in terms of performance around the center of existing techniques. We are not able to get the very accurate estimates as techniques using offline methods or accurate odometry data. This is a direct result of online-characteristic of SLAC; in terms of localisation accuracy, it will never be able to outperform offline methods as the full trace is not used.

Also, as particle filters are used, only the current position of the user is updated. All previous locations of users (which are part of the trace of a particle) remain unchanged in the update step. The only change in previous positions is caused by resampling, but this can only select a different trace as the best one and does not change actual estimated positions. By only updating the current estimation a large performance boost is achieved. If, however, the focus is to find the best user path a different method (which also updates previous positions) could be better suited.

Nevertheless, we are still able to get good estimates (average of $2.3m$), enough for room level accuracy, without using any prior information such as anchors. Only our live tests at larger environments stand out but these lower results are primarily caused by too short test runs.

6.4 Improving performance with map fusing

The SLAC system, as proposed in this thesis, runs completely on a user's device. This has the practical side effect of being privacy friendly. Nonetheless, we also concluded that short traces are insufficient for finding good estimates, especially in larger environments. This effect will be even greater when the target environment is a full sized building. Moreover, not every user will walk around the whole building. In such cases it will be very difficult to get good estimates by using a single user[3].

We can remedy this by combining data from multiple users: the process of *map fusing*. With map fusing we attempt to combine individual user's maps into a single global consistent map. See Figure 6.4.1 for an overview. Individual errors, such as mirroring errors, can potentially be removed using these kinds of methods.

A important drawback of map fusing is that information between users has to be shared. Fortunately, given the factorised approach of the algorithm, the full trace is not required for map fusing. To perform map fusing we need to be able to align individual maps of users. The individual landmark estimates are independent of each other and of the user's path if we can 'ground' the trace to some global frame.

Because the pedometer and our path loss model give real world distances we only need to rotate maps and find the starting position of the user to align them. In other words, a user's initial position, which can for example be an entrance to a building, and the individual EKFs (for each landmark) are enough to perform the map fusing. It would however still be a challenge to find a correct estimate of a user's initial position.

Map fusing offers an interesting follow-up research to the SLAC algorithm. Especially because it makes it possible to distribute the mapping of environment to multiple users. Moreover, by fusing individual EKF's the localisation error of devices can be decreased. Due to the factorised approach we do not need to share the full path of the user to make this possible; this assures that the privacy of users can be guaranteed.

[3]Assuming we do not want to force a single user to walk around the whole building for a longer period of time.

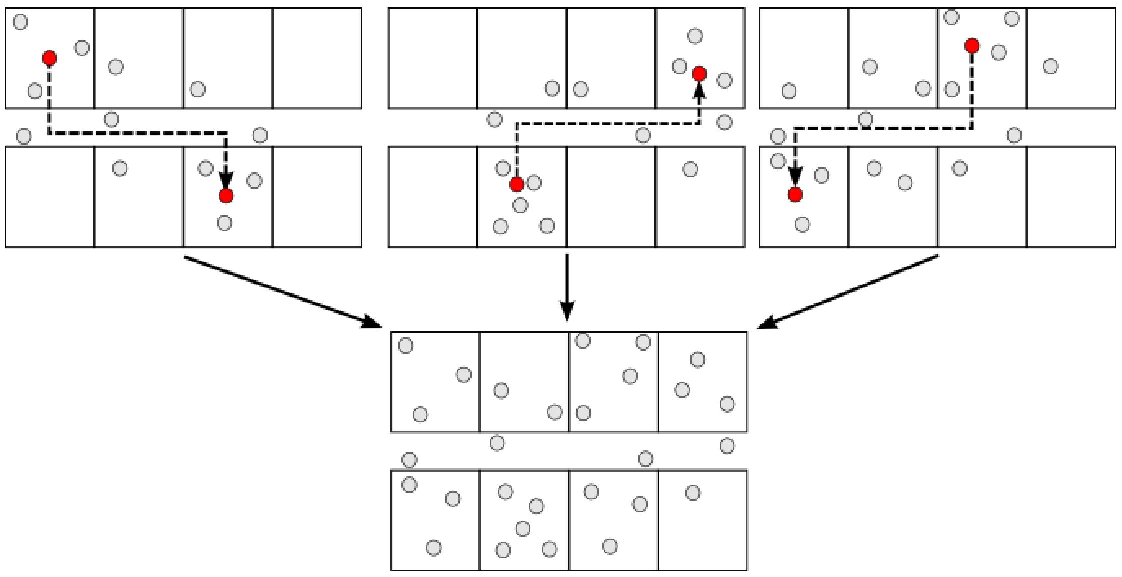


Figure 6.4.1: Overview of the idea of map fusing. Three different users (top, red dots) walk around (dashed lines) a building but their individual runs only result in a partial estimate of the environment (each device is depicted as a grey dot). By fusing these three estimates together a more complete map of the environment can be created (bottom).

6.5 Conclusion

To conclude, with SLAC we translated the simultaneous localisation and mapping problem to indoor localisation of smart spaces. Our focus did not solely lie on locating users but also on locating (smart) devices.

The traditional SLAM approach was adapted to use 1D signal strength measurements as input. These signal strengths are part of (almost) all wireless technologies used within smart spaces, and can therefore be considered a 'free' input. A motion model, using accelerometer and compass data as input, was used to make estimations of user movement. The full algorithm runs online, with the full update function computed in real time and does not require historical data to run.

Live tests, in non-trivial environments, showed that room level accuracy is indeed possible and that localisation of devices can be done very fast. This is all done locally, i.e. running on users' devices, with respect for user privacy and without using prior information of the environment.

More work is required to increase accuracy in large environments and to make the algorithm more robust for environment noise caused by walls and other objects. Existing techniques could alleviate these problems, e.g. by implementing map fusing and letting users work together.

Bibliography

Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

Aline Baggio and Koen Langendoen. Monte Carlo localization for mobile wireless sensor networks. *Ad Hoc Networks*, 6(5):718–733, July 2008.

David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

N. C. Batista, R. Melício, J. C O Matias, and J. P S Catalão. Photovoltaic and wind energy systems monitoring and building/home energy management using ZigBee devices within a smart grid. *Energy*, 49(1):306–315, 2013.

Marie Chan, Daniel Estève, Christophe Escriba, and Eric Campo. A review of smart homes- present state and future challenges. *Computer methods and programs in biomedicine*, 91(1):55–81, July 2008.

Moonok Choi, Wan-ki Park, and Ilwoo Lee. Smart Office Energy Management System Using Bluetooth Low Energy Based Beacons and a Mobile App. In *Consumer Electronics (ICCE), 2015 IEEE International Conference on*, pages 501–502, 2015.

Po-Jen Chuang and Cheng-Pei Wu. An Effective PSO-Based Node Localization Scheme for Wireless Sensor Networks. In *2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 187–194. Ieee, 2008.

Pontus Ekberg and Edith C.-H. Ngai. A distributed Swarm-Intelligent Localization for sensor networks with mobile nodes. In *2011 7th International Wireless Communications and Mobile Computing Conference*, pages 83–88. Ieee, July 2011.

Brian Ferris, Dieter Fox, and ND Lawrence. WiFi-SLAM Using Gaussian Process Latent Variable Models. In *IJCAI*, pages 2480–2485, 2007.

van Lex Gijssel and Andries Stam. ProHeal, Automated Protection and Healing Software Solutions, Project Outline Annex. Technical report, Information Technology for European Advancement (ITEA2), 2014.

GPy-authors. GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`, 2012–2014.

DM Dae-Man Han and JH Jae-Hyun Lim. Design and implementation of smart home energy management systems based on zigbee. *IEEE Transactions on Consumer Electronics*, 56(3):1417–1425, August 2010.

Guangjie Han, Huihui Xu, TQ Duong, Jinfang Jiang, and Takahiro Hara. Localization algorithms of wireless sensor networks: a survey. *Telecommunication Systems*, 52(4): 2419–2436, 2013.

Geoffrey A Hollinger and Joseph A Djugash. Tracking a Moving Target in Cluttered Environments with Ranging Radios. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1430–1435, 2008.

Geoffrey a. Hollinger, Joseph Djugash, and Sanjiv Singh. Target tracking without line of sight using range from radio. *Autonomous Robots*, 32(1):1–14, July 2011.

Lingxuan Hu and David Evans. Localization for mobile sensor networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking - MobiCom '04*, pages 45–57, New York, New York, USA, 2004. ACM Press.

Joseph Huang, David Millman, Morgan Quigley, David Stavens, Sebastian Thrun, and Alok Aggarwal. Efficient, generalized indoor WiFi GraphSLAM. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1038–1043. Ieee, May 2011.

Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *Advances in artificial life*, pages 704–720. Springer, 1995.

Neil Lawrence and Joaquin Quiñonero Candela. Local distance preservation in the GP-LVM through back constraints. In *Proceedings of the 23rd international conference on Machine learning*, pages 513–520, 2006.

Neil D. Lawrence. Gaussian Process Latent Variable Models for Visualisation of High Dimensional Data. In *Advances in Neural Information Processing Systems 16*, pages 329–336. The MIT Press, 2004.

Neil D. Lawrence. Probabilistic Non-linear Principal Component Analysis with Gaussian Process Latent Variable Models. *The Journal of Machine Learning Research*, 6:1783–1816, 2005.

Li Li and Thomas Kunz. Localization Applying An Efficient Neural Network Mapping. In *Proceedings of the 1st International ICST Conference on Autonomic Computing and Communication Systems*. Icst, 2007.

E. Menegatti, M. Danieletto, M. Mina, a. Pretto, a. Bardella, S. Zanconato, P. Zanuttigh, and a. Zanella. Autonomous discovery, localization and recognition of smart objects through WSN and image features. *2010 IEEE Globecom Workshops*, pages 1653–1657, December 2010.

Sébastien Ménigot. Pedometer in HTML5 for Firefox OS and Firefox for Android, 2014.

Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. *Proc. of 8th National Conference on Artificial Intelligence/14th Conference on Innovative Applications of Artificial Intelligence*, 68(2):593–598, 2002.

Edwin Olson, John J. Leonard, and Seth Teller. Robust range-only beacon localization. *IEEE Journal of Oceanic Engineering*, 31(4):949–958, 2006. ISSN 03649059. doi: 10.1109/JOE.2006.880386.

Ashutosh Patri and Sai Prasanna Rath. Elimination of Gaussian noise using entropy function for a RSSI based localization. In *2013 IEEE Second International Conference on Image Information Processing (ICIIP-2013)*, pages 690–694. Ieee, December 2013.

M.G. Rabbat and R.D. Nowak. Decentralized source localization and tracking [wireless sensor networks]. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 921–924. IEEE, 2004.

Carl Edward Rasmussen. Gaussian Processes in Machine Learning. In *Advanced Lectures on Machine Learning*, pages 63–71. Springer, 2006.

Carl Edward Rasmussen and Chris Williams. *Gaussian Processes for Machine Learning.* the MIT Press, 2006.

Carsten Röcker. Services and applications for smart office environments-a survey of state-of-the-art usage scenarios. In *Proceedings of the International Conference on Computer and Information Technology (ICCIT 2010), Cape Town, South Africa*, pages 1173–1189, 2010.

S T Roweis and L K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science (New York, N.Y.)*, 290(1994):2323–2326, 2000.

Scott Y Seidel and Theodore S Rappaport. 914 mhz path loss prediction model for indoor wireless communications in multi-floored building. *IEEE Transactions on Antennas and Propagation*, 40(1):207–217, 1992.

JP Sheu, WK Hu, and JC Lin. Distributed Localization Scheme for Mobile Sensor Networks. *Mobile Computing, IEEE Transactions on*, 9(4):516–526, 2010.

Marjorie Skubic, Gregory Alexander, Mihail Popescu, Marilyn Rantz, and James Keller. A smart home application to eldercare: current status and lessons learned. *Technology and health care : official journal of the European Society for Engineering and Medicine*, 17(3):183–201, January 2009.

K Sreenath, FL Lewis, and DO Popa. Simultaneous adaptive localization of a wireless sensor network. *ACM SIGMOBILE Mobile Computing and Communications Review*, 11(2):14–28, 2007.

J A Stankovic, Q Cao, T Doan, L Fang, Z He, R Kiran, S Lin, S Son, R Stoleru, and A Wood. Wireless sensor networks for in-home healthcare: Potential and challenges. In *High confidence medical device software and systems (HCMDSS) workshop*, pages 2–3, 2005.

Roy Sterrit and Dave Bustard. Autonomic Computinga Means of Achieving Dependability? In *Proceedings of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS03)*, 2003a.

Roy Sterrit and Dave Bustard. Towards an Autonomic Computing Environment. In *23rd International Workshop on Database and Expert Systems Applications. IEEE Computer Society*, pages 694–698, 2003b.

Dali Sun, Alexander Kleiner, and Thomas M. Wendt. Multi-robot range-only SLAM by active sensor nodes for urban search and rescue. In *Robocup 2008: Robot Soccer World Cup XII*, volume 5399, pages 318–330, 2009.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics.* the MIT Press, 2005.

Michael E. Tipping and Christopher M. Bishop. Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, August 1999.

Torres-González, JR A and Dios, and A Ollero. Efficient Robot-Sensor Network Distributed SEIF Range-Only SLAM. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1319–1326, 2014. ISBN 9781479936847.

Raquel Urtasun, David J. Fleet, and Neil D. Lawrence. Modeling human locomotion with topologically constrained latent variable models. In *Human Motion Understanding, Modeling, Capture and Animation. Second Workshop, Human Motion 2007, Rio de Janeiro, Brazil, October 20, 2007. Proceedings*, pages 104–118, 2007.

Jack Wang, David Fleet, and Aaron Hertzmann. Gaussian process dynamical models. In *Advances in Neural Information Processing Systems*, pages 1441–1448, 2006.

Ehsan Ullah Warriach, Tanir Ozcelebi, and Johan J Lukkien. Self- * Properties in Smart Environments: Requirements and Performance Metrics. In *Workshop Proceedings of the 10th International Conference on Intelligent Environments*, pages 194–205, 2014.

Zheng Yang, Chenshu Wu, and Yunhao Liu. Locating in fingerprint space: wireless indoor localization with little human intervention. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 269–280, 2012.

Neil Zhao. Full-featured pedometer design realized with 3-Axis digital accelerometer. *Analog Dialogue*, 44(6), 2010.

Appendix A

Particle Filters and FastSLAM

The SLAC system proposed in thesis builds upon FastSLAM (Montemerlo et al., 2002). In this appendix chapter we will explain the general idea behind FastSLAM. To do this we must first explain two other techniques which are essential components of FastSLAM: particle filters and the *(extended) Kalman filter (EKF).* Notation wise we follow mainly the definition as described by Thrun, Burgard, and Fox (2005).

A.1 (Extended) Kalman filter

The *Kalman filter* is a state estimator that makes an estimate of some unobserved variable based on noisy measurements. It is a recursive algorithm as it takes the history of measurements into account. Applied to localisation, we want to predict the new position after some motion command (the control) and new measurements of the environment. The Kalman filter assumes that the transition and observation model are linear:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{A.1.1}$$

$$z_t = C_t x_t + \delta_t \tag{A.1.2}$$

In these questions x resembles the estimate of our state and z a new measurement. A_t describes what happens to our position estimate regardless of the control. In many cases this is a identity matrix if the object of which we estimate the position does not move. On the other hand, a drone would have a transition model that incorporates influences of wind. B_t describes how a specific control u changes the state from $t-1$ to t. ϵ_t describes Gaussian process noise with covariance R_t. C_t describes the mapping from a state to an observation which definition depends on the context.

These two linear models can be incorporated in the definition of the multivariate normal distribution[1]. We then have the probability of a position given the control and the previous estimate:

$$\begin{aligned} p(x_t|x_{t-1}, u_t) =& \det(2\pi R_t)^{-\frac{1}{2}} \\ & \exp\Big(-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t)\Big) \end{aligned} \tag{A.1.3}$$

[1] $p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\Big(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\Big)$ with μ the mean and Σ the covariance matrix.

Here $A_t x_{t-1} + B_t u_t$ describes the mean of the distribution (as it is the estimated new position given our model) and R_t the covariance matrix. The probability of a certain measurement is then described by:

$$p(z_t|x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\Big(-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1}(z_t - C_t x_t)\Big) \tag{A.1.4}$$

with $C_t x_t$ the mean and Q_t the covariance.

The Kalman filter computes a weighted average between the prediction and the observation based on the certainty of these two. The resulting belief wil lie closest to the input distribution with the highest certainty.

The belief of the prediction (i.e. our state estimate without incorporating the measurement), $\overline{bel}$, is based on the previous belief and the control:

$$\overline{bel}(x_t) = \int p(x_t|x_{t-1}, u_t)\ bel(x_{t-1})\ dx_{t-1} \tag{A.1.5}$$

The final belief of our state is based on the predicted belief and the measurement:

$$bel(x_t) = \eta\ p(z_t|x_t)\ \overline{bel}(x_t) \tag{A.1.6}$$

See Thrun et al. (2005, chap. 3.2.4) for a mathematical derivation of these equations.

A.1.1 Non-linear functions

The transition and observation model we described earlier are linear models. Unfortunately, many real world situations are not linear. Consider for example the translation of a position to a distance measurement, this is a non-linear transformation. To counter this we can replace our linear transformations with non-linear functions:

$$x_t = g(x_{t-1}, u_t) + \epsilon_t \tag{A.1.7}$$

$$z_t = h(x_t) + \delta_t \tag{A.1.8}$$

These non-linear functions do however leed to non-Gaussian distributions. The *Extended Kalman Filter* addresses this by linearising the non-linear functions at the mean of the input distribution. This linearisation is an approximation but offers fairly good performance if the values are not far from the linearisation point; i.e. the higher the uncertainty of the Gaussian the higher the error.

The linearisation is performed through the use of first order Taylor expansion. Using the linearisation our new probability distributions for the postion and the measurements then become:

$$\begin{aligned} p(x_t|u_t, x_{t-1}) \approx & \det(2\pi R_t)^{-\frac{1}{2}} \exp\Big(-\frac{1}{2}(x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})^T \\ & R_t^{-1}(g(u_t, \mu_{t-1} - G_t(x_{t-1} - \mu_{t-1})\Big) \end{aligned} \tag{A.1.9}$$

$$\begin{aligned} p(z_t|x_t) = & \det(2\pi Q_t)^{-\frac{1}{2}} \exp\Big(-\frac{1}{2}(z_t - h(\bar{\mu}_t) - H_t(x_t - \bar{\mu}_t))^T \\ & Q_t^{-1}(z_t - h(\bar{\mu}_t) - H_t(x_t - \bar{\mu}_t))\Big) \end{aligned} \tag{A.1.10}$$

A.1.2 Updating the EKF

To perform state estimation the extended Kalman filter continuously updates its belief of the system when new measurements and controls come in. This is done through the Kalman update, as shown in Algorithm A.1. The algorithm shows the update for the EKF but is, without the linearisation, very similar to the update function of the regular Kalman filter. The Kalman gain (K_t) is used as the weighing factor: the larger the Kalman gain, the more we 'distrust' our current estimate and the larger the actual update of the estimate will be.

Algorithm A.1 Extended Kalman filter algorithm. The filter tries to estimate some state modelled through μ. An update is performed given a new control u and measurement z.

```
1: function EKF($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)

2:     $\bar{\mu}_t = g(\mu_{t-1}, u_t)$
3:     $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$

4:     $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^- 1$

5:     $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6:     $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$

7:     return $\mu_t, \Sigma_t$
8: end function
```

A.2 Particle Filters

Particle filters are a type of non-parametric filters to approximate a posterior distribution. They are sometimes also called as *Sequential Monte Carlo* methods. Particle filters can be seen as a probabilistic implementation of Darwin's *Survival of the Fittest* and are a type of *genetic algorithms.* Here we describe shortly the general idea of particle filters and their inner workings.

The basis of a particle filter is a set of particles. Each particle represents, at a given time t, a concrete possible state of the concept that is being modelled. A single particle can be seen as a hypothesis about the world at a given moment in time. More formally, each particle is a sample of the posterior distribution. The amount of particles is often large and fixed but, in some cases, can also be a function of some other parameter.

In robotics and indoor localisation, each particle is often a sample of the distribution of the current expected location of a robot or person. Each particle represents a valid location and the total particle space describes the range of the estimate. In the remainder of this section we will use the term 'robot' to describe the actor in the particle filter.

A.2.1 Basic algorithm

The particle filter is a recursive algorithm: it computes its new posterior estimate based on the previous state (χ_{t-1}) and some additional input. This additional input consist of the robot's control (u_t) and its sensor measurements (z_t). This combination is crucial;

the control describes where the robot should go, the sensor readings describe roughly whether this action has succeeded.

Algorithm A.2 Particle Filter algorithm. Input is the particle set χ, the control u, the measurement z and the desired number of particles M.

```
1: function ParticleFilter($\chi_{t-1}, u_t, z_t, M$)
2:     $\bar{\chi}_t = \chi_t = \emptyset$

3:     for m = 1 to M do                                  ▷ Sampling step
4:         Retrieve $x_{t-1}^{[m]}$ from $\chi_{t-1}$
5:         Sample $x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$
6:         $w_t^{[m]} = p(z_t|x_t^{[m]})$                  ▷ Importance factor
7:         $\bar{\chi}_t = \{\chi_t + \langle x_t^{[m]}, w_t^{[m]}\}$
8:     end for

9:     for m = 1 to M do                                  ▷ Resampling step
10:        Draw $i$ with probability $\propto w_t^{[i]}$
11:        $\chi_t = \chi_t + x_t^{[i]}$
12:    end for

13:    return $\chi_t$
14: end function
```

The most basic version of the particle filter algorithm consists of two important steps: the *sampling* and *resampling step.* The algorithm is displayed in Algorithm A.2. The algorithm starts by generating M new samples as part of the sampling step. These samples are drawn from a distribution based on the previous particle $x_{t-1}^{[m]}$ and the robot's control u_t where m denotes the current particle. For each new sample an importance factor is calculated (line 5). This importance factor is the probability of the measurement z_t given the new sample. This factor gives an indication of how good the sample can explain the sensor reading. The general idea is that a good sample will match the sensor reading better.

After the sampling step we end with a list of M particles and an importance factor per particle. In the resampling step (line 8) we draw particles from this temporary set ($\bar{\chi}_t$), with replacement, and proportional to the probability defined by the importance factor; i.e. particles with a good estimate have a higher probability of ending up in the new particle set. When the resampling step is complete we end up with a new set of particles (with possible duplicates) that describes the new estimate of the robot's position.

A.2.2 Design considerations

There are a few properties of particle filters that have to be taken in to account as these can degrade the performance of the filter (Thrun et al., 2005). We highlight some of these here shortly and describe how these could affect localisation.

Density extraction: The estimate of the robot's position is a continuous distribution over the state space. However, we only describe this estimate with a finite amount of particles. So our belief of the location is a discrete approximation. Multiple methods exist to overcome this problem. For localisation a easy method is just averaging the particles; this is particularly useful if there is limited processing power available.

Sampling variance: As we take finite random samples from a probability distribution this introduces errors. This variability is called the sample variance. The sample variance can be minimised by increasing the amount of particles at the expense of increasing complexity.

Loss of diversity: The resample step favours particles with a high importance factor and these can be selected multiple times (due to the replacement). This has as a result that the diversity of particles can drop as the algorithm progresses. While the variance between particles will also drop, the variance between the particles and the true belief can increase due to this. This loss of diversity can be addressed by different sampling methods.

Particle deprivation: When there are no particles near the correct state we speak of the *particle deprivation problem*. This problem occurs most often when the amount of particles is small and is a result of the random sampling. A specific series of random draws can result in particle deprivation. However, the larger the amount of particles the less likely this is to occur.

A.3 FastSLAM

As stated before, *FastSLAM* is a particle filter implementation of SLAM. Initially, FastSLAM is defined to solve the *full* SLAM problem: given all the sensor readings and robot controls (from $1 : t$), what is the full path and map of the environment? The map is represented by a set of landmarks. These landmarks can be anything; in the case of indoor localisation they are often devices.

The full SLAM problem introduces a conditional independence that the FastSLAM algorithm utilises, using *Rao-Blackwellised Particle filters*, to increase performance: given the robot path, the location of the landmarks are independent of each other and can be estimated separately. See also Figure A.3.1 for a visual explanation. In order to do this we factorise the full SLAM posterior[2]:

$$\begin{aligned}
p(x_{0:t}, m_{1:M} | z_{1:t}, u_{1:t}) &= \text{path posterior} \times \text{map posterior} \\
&= p(x_{0:t}|z_{1:t}, u_{1:t})\; p(m_{1:M}|x_{0:t}, z_{1:t}, u_{1:t}) && \text{(A.3.1)} \\
&= p(x_{0:t}|z_{1:t}, u_{1:t})\; p(m_{1:M}|x_{0:t}, z_{1:t}) && \text{(A.3.2)} \\
&= p(x_{0:t}|z_{1:t}, u_{1:t}) \prod_{i=1}^{M} p(m_i|x_{0:t}, z_{1:t}) && \text{(A.3.3)}
\end{aligned}$$

Here $x_{0:t}$ describes the robot's path, $m_{1:M}$ the landmarks, $z_{1:t}$ the sensor measurements and $u_{1:t}$ the controls. Note that we omitted the controls $u_{1:t}$ in the estimation of the map

[2]This definition differs slightly from the one in Thrun et al. (2005). Here we assume that there is no data association problem so the correspondence is not part of the posterior.

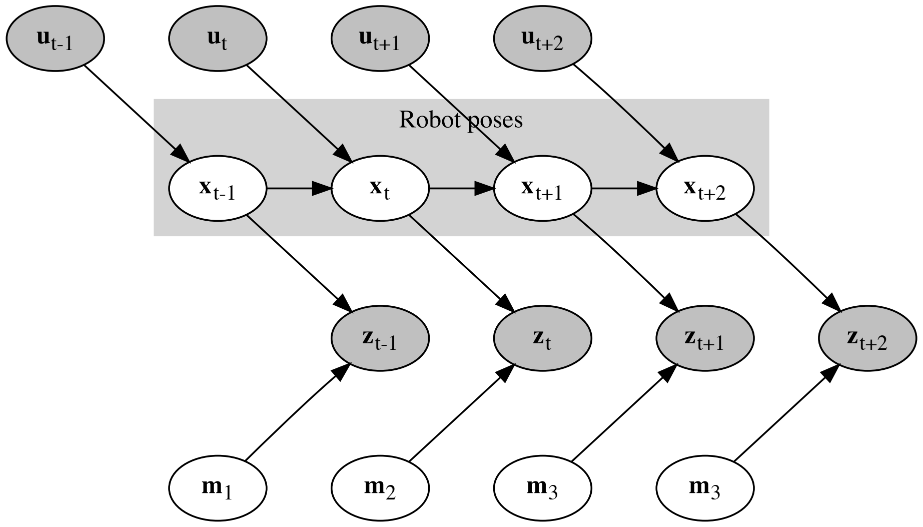


Figure A.3.1: Bayesian network representation of the FastSLAM algorithm. Grey nodes are observed variables, white nodes are latent. Given the robot path there is no other path between landmarks; i.e. given the path the landmark locations are independent.

(Equation A.3.2); given the robot's path the location of the landmarks are independent of the controls.

By factorising we transformed a highly dimensional problem into two separate problems: estimating the path and the landmarks. FastSLAM uses a particle filter to estimate the robot path. The mapping problem, which is easily computable given the robots path, is factored in separate problems; one for each landmark. The individual landmark locations can be estimated using a low-dimensional EKF. This is in contrast to other SLAM methods who usually have a joint estimate of all landmarks.

Even though we make an estimate about the whole path of the robot in Equation A.3.3, we will see that FastSLAM can also be used for *online SLAM*: instead of estimating the whole path we only want estimate the current position. This is made possible due to the definition of the particle filter: the estimate is only dependent on the previous estimate and the whole path is not required.

A.3.1 Algorithm in more detail

The basic FastSLAM algorithm (Montemerlo et al., 2002) is closely related to the particle filter algorithm. As can be seen in Algorithm A.3, the two main steps of the particle filter are present: the sampling and resampling step. In the sampling step particles are updated based on the robot's control. In the resampling step new particles are drawn with replacement from the generated particle set ($\bar{Y}_t$).

The main addition in comparison to the particle filter algorithm is the addition of an

Algorithm A.3 FastSLAM algorithm. Input is the particle set Y, the control u, the set of measurements Z and the desired number of particles M.

```
1: function FASTSLAM($Y_{t-1}, u_t, Z_t, M$)
2:     $\bar{Y}_t = Y_t = \emptyset$

3:     for $m = 1$ to $M$ do
4:         Retrieve a pose $x_{t-1}^{[m]}$ from particle set $Y_{t-1}$
5:         Sample $x_t^{[m]} \sim p(x_t | x_{t-1}^{[m]}, u_t)$                    ▷ Sample step

6:         for $i = 1$ to $|Z_t|$ do                                   ▷ Landmark update step
7:             Identify correspondance $j$
8:             if j is new then
9:                 Initialise $\mu_{j,t}^{[m]}$ and covariance $\Sigma_{j,t}^{[k]}$ given $z_t^i$.
10:            else
11:                Update mean $\mu_{j,t}^{[m]}$ and covariance $\Sigma_{j,t}^{[k]}$ given $z_t^i$.
12:            end if
13:        end for

14:        Calculate importance weight $w^{[m]}$
15:        $\bar{Y}_t = \bar{Y}_t + \{x_t^{[m]}, w_t^{[m]}, \text{landmarks}\}$    ▷ Including unobserved landmarks
16:    end for

17:    for m = 1 to M do                                               ▷ Resampling step
18:        draw $i$ with probability $\propto w_t^{[k]}$
19:        $Y_t = Y_t + \{x_t^{[k]}, \text{landmarks}\}$
20:    end for

21:    return $Y_t$
22: end function
```

additional step in which the estimates of the observed landmarks (i.e. the devices) are updated. This is necessarily for a robot to generate a map. First we need to correspond the observation z_t^i to a specific landmark (i.e. the *data association problem*). When this link is established we can update our estimate of that landmarks position. Finding the correspondence is trivial when data association is known; e.g. with uniquely identifiable landmarks.

Note that the landmarks and the robot pose are separate and not part of a combined state space; i.e. we do not sample over the robot poses and the landmarks combined. This is the most important aspect of the FastSLAM algorithm. As we explained before, by factorising the problem we can minimise the amount of particles that are needed due to each particle representing a smaller dimension.

At time t, a single particle in the FastSLAM algorithm can be described as:

$$Y_t^{[m]} = \left\langle x_t^{[m]}, \langle \mu_{1,t}^{[m]}, \Sigma_{1,t}^{[m]} \rangle, \ldots, \langle \mu_{N,t}^{[m]}, \Sigma_{N,t}^{[m]} \rangle \right\rangle \tag{A.3.4}$$

with $m \in M$ describing the particle index, M the total amount of particles and N the amount of landmarks. $x_t^{[m]}$ describes the estimated robot position in 3 dimensions

(x, y, θ). Each $\mu_{j,t}^{[m]}, \Sigma_{j,t}^{[m]}$ are the mean and variance of a Gaussian describing the estimate of a single landmark l with $j \in N$.

Calculating the posterior at time t boils down to generating a new particle set Y_t from the previous one Y_{t-1}. For this we execute the three main steps of the FastSLAM algorithm (the steps refer to Algorithm A.3):

Sample (or prediction) step: Using the robot control u_t a new pose is sampled according to the motion model:

$$x_t^{[m]} \sim p(x_t | x_{t-1}^{[m]}, u_t) \tag{A.3.5}$$

The new pose is always computed using the previous estimate from the same particle, $x_{t-1}^{[m]}$.

Landmark update step: For each observed landmark we must update the estimate and therefore update the mean $\mu_{j,t-1}^{[m]}$ and covariance $\Sigma_{j,t-1}^{[m]}$ using the EKF update function. If a landmark is not observed the estimated position remains unchanged. When a new (previous unseen) landmark is observed we initialise the EKF with the current measurement.

Resampling (or correction) step: In the resampling step we draw (with replacement) a new set of particles with probabilities proportional to the importance weights. This resampling is required as the measurements are not embedded in the distribution sampled in the sample step. From the sample step, we obtain a distribution given only our motion model. This is not equal to our target distribution which favours positions based on the measurements. By sampling particles based on the weight, which is based on the measurements, we correct for this mismatch.

Appendix B

Random variables and distributions

Consider a simple example such as rolling a fair dice d_1: there are six distinct outcomes and each outcome has an equal probability (namely $\frac{1}{6}$). We define this set of outcomes as Ω; this is a discrete distribution. Discrete distributions can only have a finite (which is clearly the case in our example) or a countable infinite number of values.

Now we add a second dice, d_2, and we want to reason about the sum of the two dices: e.g. what is the probability of an even value? Note that we are now not interested anymore in the individual outcomes but in events. The outcome $(n_1 = 1, n_2 = 3)$ is in our model the same event as $(n_1 = 1, n_2 = 1)$ as the sum is even in both cases.

For this we need a mapping from the set of outcomes to a different variable we wish to model. We can define a measurable function, X, that describes this relation, which is called a *random* (or *stochastic*) variable. Formally, a random variable can be defined as a function

$$X : \Omega \to \mathbb{R} \tag{B.0.1}$$

in which Ω is the set of possible outcomes. Note that $\mathbb{R}$ could also be replaced by a different type of set (but is often $\mathbb{R}$). The application of this to our dice example is simple:

$$X(n_1, n_2) = \begin{cases} 1 & \text{if } (n_1 + n_2) \bmod 2 = 0 \\ 0 & \text{otherwise} \end{cases} \tag{B.0.2}$$

We map all the different outcomes (different values of n_1 and n_2) to numerical values. Note that each individual dice role can also be described by a random variable which simply uses the value of the dice.

B.1 Types of distributions

In our single-dice problem the probabilities of each event was trivial as all were equal. The two-dice situation has a different probability distribution since not every event has the same likelihood. In the next sections we will describe how we can define these distributions.

B.1.1 Discrete probability distributions

In the case of a discrete random variable we can define a probability distribution P that maps events (subsets of outcomes) to some value:

$$P(\Omega) := \{A \subseteq \Omega\} \rightarrow \mathbb{R} \tag{B.1.1}$$

with Ω beging the set of possible outcomes. For discrete random variables this distribution is called a *probability mass function* and is usually described as (for a random variable X):

$$\begin{aligned} f_X(x) = P(X = x) &= P(\{\omega \in \Omega \mid X(\omega) = x\}) \\ &= \sum_{X(\omega)=x} P(\omega) \end{aligned} \tag{B.1.2}$$

which states that we compute the probability of all the outcomes ($\omega \in \Omega$) that result in the given event x. The distribution has the following characteristics:

$$\begin{aligned} P(\Omega) &= 1 && \text{(B.1.3)} \\ \forall A \subseteq \Omega,\ P(A) &\geq 0 && \text{(B.1.4)} \\ A \cap B = \emptyset \Rightarrow P(A \cup B) &= P(A) + P(B) && \text{(B.1.5)} \end{aligned}$$

which can be explained as: all the possible outcomes lay in Ω, all probabilities are non-negative and outcomes which are independent can be simply added. This last characteristic is, possibly, the least intuitive but can be explained with a simple example: Consider our two dice example. As individual throws are independent we can simply say: $P(even) = P(2) + P(4) + P(6) + P(8) + P(12)$.

The expected value or mean of a discrete random variable can easily be defined as:

$$E[X] = \sum_{i=1}^{n} x_i f_X(x_i) \tag{B.1.6}$$

B.1.2 Continous probability distributions

When Ω is a continuous set, the probabilities are not assigned to values but to intervals. Namely, as the space of values is infinite the probability of a single value is practically zero. To compute the probability of an interval, an integral can be used over the range of that particular interval. So, given a probability distribution function $f_X(x)$, the probability that an outcome x lays within an interval $[a, b]$ can be defined as:

$$P(a \leq x \leq b) = \int_a^b f_X(x)dx \tag{B.1.7}$$

with the following characteristics:

$$\begin{aligned} \forall x \in \mathbb{R},\ f_X(x) &\geq 0 && \text{(B.1.8)} \\ \int_{-\infty}^{\infty} f_X(x)dx &= 1 && \text{(B.1.9)} \\ A \cap B = \emptyset \Rightarrow P(A \cup B) &= P(A) + P(B) && \text{(B.1.10)} \end{aligned}$$

where A and B can be seen as two non-overlapping intervals. Note that these characteristics are analogous to those of discrete random variables. If $f_X(x) : \mathbb{R} \rightarrow \mathbb{R}$ then f_X is called a *probability density function (pdf)* of X. The expected value or mean of a continuous probability distribution can be computed with:

$$E[X] = \int_{-\infty}^{\infty} x f_X(x)\, dx \tag{B.1.11}$$

B.1.3 Gaussian distribution

A special variant of the continuous probability distribution is the *gaussian distribution* (or normal distribution). For a variable X to be gaussian distributed, its probability density function is defined by:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{B.1.12}$$

with μ the mean and σ describing the standard deviation of the distribution. The constant $\frac{1}{\sigma\sqrt{2\pi}}$ assures that the area under the curve is equal to one. When μ is 0 and σ is 1 we speak of a *standard normal* distribution.

B.1.4 Joint probability distributions

Until this point we only considered univariate distributions; i.e. a probability distribution of only one random variable or a random variable that combines multiple variables in to one (as with the example of predicting an even result from two dice). When we have more than one random variable, the distribution becomes a *multivariate* or *joint probability distribution.* Such a set of random variables is usually called a *random vector.*

B.2 Stochastic (or random) process

Apart from looking at individual random variables, we can also investigate series. A *random* (or *stochastic*) *process* is, in its most simple definition, a collection of random variables. It models the development of a system in which the transfer between states is non-deterministic (therefore 'random') and can be defined as:

$$\{X_t \mid t \in T\} \tag{B.2.1}$$

with T being an ordered set, $T \subseteq [0, \infty)$, which in most cases represents time or space[1] and each X_t begin a random variable. We can use a random process to model some system that changes through time in a non-predictable way. This is in contrast to a deterministic system in which the steps are known given knowledge about the initial state. An example of a random process is the stock price of a company observed over time.

A random process can be continuous or discrete; e.g. when time is used, the process can be continuous (it can take any value within the time interval) or discrete (days, minutes, hours, etc.). Regardless of the type of the process, the random variables the process consists of can also be continuous or discrete; i.e. a continuous processes can consist of discrete variables and vice versa. A random process can have many different

[1]For convenience we will, in these descriptions, describe T as time.

outcomes, defined by its random factors. Each outcome is a function of time and is called a *sample function* (or possibly more intuitive: a *sample path*).

To say something about the state of a random process at a specific point in time t we can calculate the mean of the random process. The mean is a function of time:

$$\mu_X(t) = E[X(t)] \tag{B.2.2}$$

In other words: the mean of a process X at time t is the expected value of the random variable at that specific point in time.

B.2.1 Simple example: plant growth

Consider modelling the growth of a simple plant after 10 days (i.e. a discrete interval of $[0, 10]$). Lets say that at t_0 our plant has a height of 1cm and, in our toy world, the daily growth is described by a random variable G from a uniform distribution $\mathcal{U}[1, 3]$[2]. We can define a random process with:

$$\{X_t \mid t \in [0, 1, \ldots, 10]\} \tag{B.2.3}$$

$$X_t = 1 + Gt \tag{B.2.4}$$

From this follows automatically that $X_0 = 1$. The randomness of our process is defined by G, so for every $g \in \mathcal{U}[1, 3]$ we obtain a sample function for X_t in the form of: $f(t) = 1 + gt$. Using the definition of the probability density function of an uniform distribution[3] we can define the function of our random variable G:

$$f_G(x) = \begin{cases} \frac{1}{3-1} & \text{if } 1 \le x \le 3 \\ 0 & \text{otherwise} \end{cases} \tag{B.2.5}$$

Using the pdf we calculate the expected height of our plant at t_{10}:

$$\begin{aligned} E[X_t] &= 1 + E[Gt] && \text{(B.2.6)} \\ E[X_{10}] &= 1 + E[10G] && \text{(B.2.7)} \\ &= 1 + 10E[G] \\ &= 1 + 10 \int_1^3 g f_G(g)\, dg \\ &= 1 + 10 \int_1^3 \frac{1}{2} g\, dg \\ &= 1 + 10[\frac{1}{4} g^2]_1^3 \\ &= 1 + 10[2.25 - 0.25] \\ &= 21 \end{aligned}$$

[2]Clearly, this is not a good model of real plant growth.

[3]The pdf of a uniform distribution $\mathcal{U}[a, b]$ is defined as $pdf(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \le x \le b \\ 0 & \text{otherwise} \end{cases}$

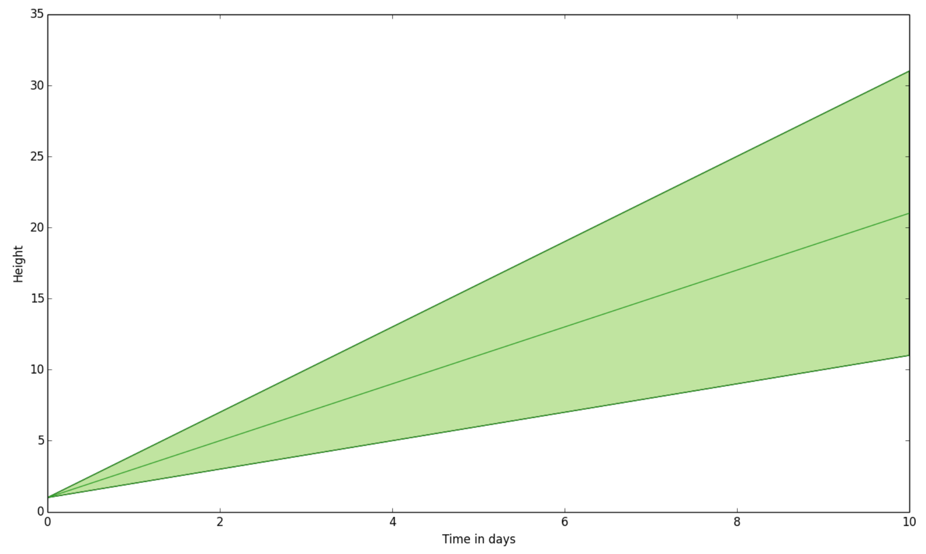


Figure B.2.1: Visualisation of the plant growth model. The great coloured area models all possible heights at a given time. The center line gives the expected value at each day given the uniform growth distribution.

B.3 Relations between variables

Using the mean of a random process we can express something about the state of the process at a specific time or space. However, we cannot say anything about the relation between two points, e.g. X_1 and X_2. For this we can define the correlation (ρ) and covariance (cov) between two variables with:

$$\begin{aligned} \rho(X_1, X_2) &= \frac{E[(X_1 - \mu_1)(X_2 - \mu_2)]}{\sigma_1 \sigma_2} \qquad \text{(B.3.1)} \\ &= \frac{E[(X_1 - E[X_1])(X_2 - E[X_2])]}{\sigma_1 \sigma_2} \\ cov(X_1, X_2) &= E[(X_1 - E[X_1])(X_2 - E[X_2]) \qquad \text{(B.3.2)} \\ (&= E[X_1 X_2] - E[X_1]E[X_2], \text{ alternate form}) \end{aligned}$$

The covariance function is an important property of a special form of random proceses, namely *Gaussian processes*. We will explain these processes in a later section. In many cases, including Gaussian processes, we are interested in the covariance between one variable and all others. For this we introduce a new notation for covariances: the covariance matrix, $\mathbf{\Sigma}$. The covariance matrix is defined as an $n \times n$ matrix with elements:

$$\mathbf{\Sigma}_{ij} = cov(X_i, X_j) \qquad \text{(B.3.3)}$$

Appendix C

Gaussian Processes

A random process, a collection of random variables, is said to be a *Gaussian process (GP)*[1] if any finite number of these variables have a joint Gaussian distribution; i.e. the relation between variables follows a Gaussian distribution, this says something about the smoothness of functions generated by these processes.

As we will shift to more complex situations (and input data) we will switch notations from from X_t to a more general $f(\mathbf{x})$ in which $\mathbf{x}$ is the input vector. Mathematically, a Gaussian process f is defined by its mean (m) and covariance function (the kernel, k), covariance functions are valid mercer kernels:

$$f \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}_i, \mathbf{x}_j)) \tag{C.0.1}$$

This states that f is distributed as a gaussian process. Here m and k are functions, in contrast to single values (univariate) or vectors and matrices (multivariate) with Gaussian distributions. So, a GP can be seen as a generalisation of a gaussian distribution on a random vector. A GP is a non-parametric model as the parameters of the model (the values of the mean and covariance function) are not known beforehand and are derived from data.

A multivariate Gaussian distribution is defined on a random vector. Each element in the vector denotes a random variable; i.e. the variables are indexed by their position. For a GP there is no such index. Instead, we have $\mathbf{x}$ which is used to indicate the random variables. For each $\mathbf{x}_i$ there is a random variable $f(\mathbf{x}_i)$ which is the value of the GP f at that location.

The mean and covariance functions are described as (analogous to B.2.2 and B.3.2):

$$m(\mathbf{x}) = E[f(\mathbf{x})] \; (= 0, \text{often used for simplicity}) \tag{C.0.2}$$

$$cov(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j) = E[(f(\mathbf{x}_i) - \mu(\mathbf{x}_i))(f(\mathbf{x}_j) - \mu(\mathbf{x}_j))] \tag{C.0.3}$$

For simplicity, the mean function $m(x)$ is usually taken to be zero, e.g. $m(\mathbf{x}) = \mathbf{0}$ where $\mathbf{0}$ is a zero-vector. This usually does not limit the model as predictions do not have to have a zero-mean. A zero mean function can be a characteristic of the data or achieved by preprocessing. The covariance is an important aspect of GPs: a GP

[1]We follow here the definitions as described by Rasmussen and Williams (2006); Rasmussen (2006); Barber (2012).

requires that the covariance between two function values $(f(\mathbf{x}_i), f(\mathbf{x}_j))$ depends on the covariance of the input values $(\mathbf{x}_i, \mathbf{x}_j)$:

$$cov(f(\mathbf{x}_i), f(\mathbf{x}_j)) = k(\mathbf{x}_i, \mathbf{x}_j) \tag{C.0.4}$$

Here k is the kernel function, we will explain in Section C.3 how to choose this function.

C.1 Noisy processes

In many real-world examples, the function values are not known directly due to noise. This can be the result of measurement noise but can also be a characteristic of the system or caused by external factors (such as noise in wireless signals). So instead of observing $f(\mathbf{x})$ we observe:

$$y_i = f(\mathbf{x}_i) + \epsilon \tag{C.1.1}$$

where each $\mathbf{x}_i$ is an input sample (from $\mathbb{R}^d$, with d the dimension) and y_i the target or observation. ϵ models the noise and describing the noise variance. This noise also changes the definition of the GP:

$$f \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}_i, \mathbf{x}_j)) \tag{C.1.2}$$

$$cov(y_i, y_j) = k(\mathbf{x}_i, \mathbf{x}_j) + \sigma_n^2 \delta_{ij} \tag{C.1.3}$$

$$y \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}_i, \mathbf{x}_j) + \sigma_n^2 \delta_{ij}) \tag{C.1.4}$$

here δ_{ij} is the Kronecker's delta (1 iff $i = j$). So, for identical points ($i = j$), the covariance function is defined by the signal covariance and the noise covariance. This only applies to identical points as the noise is assumed to be independent.

C.2 Prediction using Gaussian processes

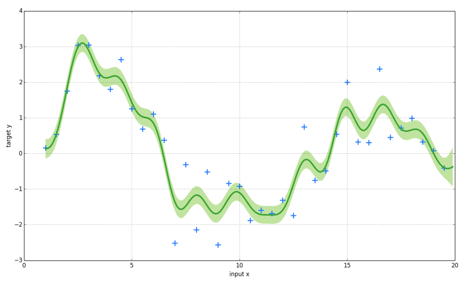
Here we consider regression (as opposed to classification) on noisy data. Given training data D consisting of n input-output pairs $(\mathbf{x}_i, y_i)$ with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, we define the whole training data as $D = \{\mathbf{X}, \mathbf{y}\}$ with $\mathbf{X}$ the matrix of input values and $\mathbf{y}$ the observations. $\mathbf{X}$ has dimensions $n \times d$ and $\mathbf{y}$ is a vector of $n \times 1$. We define the covariance matrix on $\mathbf{y}$ with

$$\mathbf{\Sigma}(\mathbf{y}) = K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I} \tag{C.2.1}$$

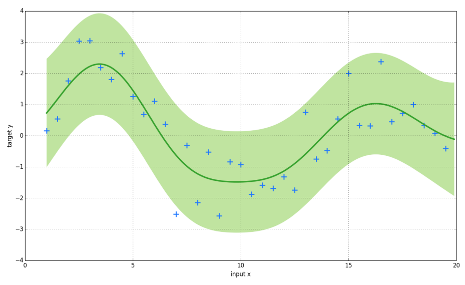
where $\mathbf{I}$ is the identity matrix and K the kernel on the input values. In other words, the covariance matrix is an $n \times n$ matrix with values as defined in Equation C.1.3. For notational simplicity we will use $\mathbf{\Sigma}$ to describe the matrix (as opposed to $\mathbf{\Sigma}(\mathbf{y})$). Using our covariance matrix we can describe our process as:

$$p(\mathbf{y}|\mathbf{X}) \sim \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{\Sigma}) \tag{C.2.2}$$

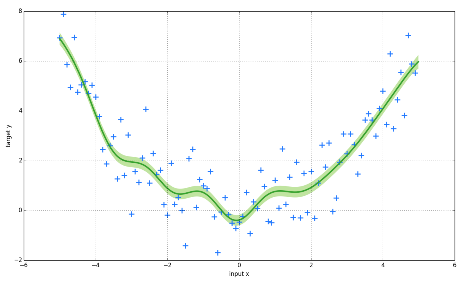
Now we introduce a new data point $(\mathbf{x}_*, y_*)$ where $\mathbf{x}_*$ is observed and we want to predict f_* (short for $f(\mathbf{x}_*)$). Note that we here focus on the *function value* instead of a



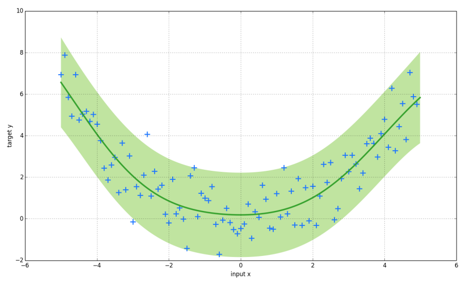
(a) GP before optimisation



(b) GP after optimisation



(c) GP before optimisation



(d) GP after optimisation

Figure C.1.1: Application of a zero-mean, squared-exponential kernel GP on a random dataset (blue plusses) generated using $y = 2\sin(\frac{1}{2}x) + \epsilon$ (figures a, b) and $y = \frac{1}{4}x^2 + \epsilon$ (figures c, d) where $\epsilon \sim \mathcal{N}(0, 1)$. The dark line shows the posterior mean (sampled using very small intervals), the green area the posterior variance. In (a) and (c) no optimisation of the hyperparameters has been performed, its clearly visible that, although the model follows roughly the structure of the underlying function, it captures to much of the noise in the data. In (b) and (d) the parameters have been optimised, resulting in a better model of the mean and variance. Figures have been generated using the Python *pyGPs* package.

noisy observation; using our data consisting of noisy observations we aim to predict the clean signal given some new input $\mathbf{x}_*$. We can write a joint distribution on our train data and the new point:

$$p(\mathbf{y}, f_*|\mathbf{X}, \mathbf{x}_*) \sim \mathcal{N}(\mathbf{y}, f_*|\mathbf{0}, \mathbf{\Sigma}^+) \tag{C.2.3}$$

$$\mathbf{\Sigma}^+ = \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) + \sigma_n^2\mathbf{I} & K(\mathbf{X}, \mathbf{x}_*) \\ K(\mathbf{x}_*, \mathbf{X}) & K(\mathbf{x}_*\mathbf{x}_*) \end{bmatrix} \tag{C.2.4}$$

where $\mathbf{\Sigma}^+$ is the extended covariance matrix, $K(\mathbf{x}_*, \mathbf{X}) = K(\mathbf{X}, \mathbf{x}_*)^T$ and $\mathbf{0}$ a zero-vector with length $n+1$. Note that for a non-noisy process the term $\sigma_n^2\mathbf{I}$ can be simply omitted.

We are however interested in the conditional probability of f_* given the data. For this we must convert the joint distribution (C.2.4) to by conditioning it using the theorem:

$$\begin{aligned} &p(\mathbf{x}, \mathbf{y}) \sim \mathcal{N}\left(\mathbf{a}, \mathbf{b} \,\middle|\, \begin{bmatrix} A & C \\ C^T & B \end{bmatrix}\right) \implies \\ &p(\mathbf{x}|\mathbf{y}) \sim \mathcal{N}(\mathbf{a} + CB^{-1}(\mathbf{y} - \mathbf{b}), A - CB^{-1}C^T) \end{aligned} \tag{C.2.5}$$

We can now compute the predictive distribution for f_* given Equations C.2.3 and C.2.5:

$$p(f_*|x_*, D) = p(f_*|x_*, \mathbf{X}, \mathbf{y}) \sim \mathcal{N}(f_*|m_D, k_D)) \tag{C.2.6}$$

$$m_D = K(\mathbf{x}_*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}]^{-1} \mathbf{y}, \tag{C.2.7}$$

$$k_D = K(\mathbf{x}_*, \mathbf{x}_*) - K(\mathbf{x}_*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}]^{-1} K(\mathbf{X}, \mathbf{x}_*) \tag{C.2.8}$$

The covariance k_D depends not on the observed targets ($\mathbf{y}$) but only on the prior variance ($K(\mathbf{x}_*, \mathbf{x}_*)$) subtracted by a positive term that is dependent on the training data; i.e. the training data lowers the variance as it gives us information about the process. Even though we have used a zero-mean function for the GP, the mean of the posterior is not necessarily zero (due to the definition of m_D). Note that we followed Equation C.2.5 and that our joint probability (Equation C.2.3) has a zero mean. Predicting a set of points can be done by replacing f_* by a vector $\mathbf{f}_*$.

C.3 Training a Gaussian process

As we saw a GP is defined by its mean and covariance function and, in the case of a noisy process, also the noise variance. The process of training a GP consists of defining these functions based on the training data using two steps:

1. **Model selection**: choosing the functional form of the mean and covariance functions.
2. **Adapting hyperparameters**: optimising the parameters of the functions given the training data.

C.3.1 Model selection

In GP's we have to define two priors on the model: the mean and covariance functions. The mean function is usually defined as a zero-vector. The choice of the covariance function is however of great importance as it defines the interaction between consecutive datapoints. An often used covariance function is the *Gaussian* or *squared-exponential kernel* which, in its most simple form, can be defined as:

$$k(x_i, x_j) = \exp(-|x_i - x_j|^2) \tag{C.3.1}$$

However, most of the time extra parameters are added to tune the kernel:

$$k(x_i, x_j) = \sigma_f^2 \exp\left(-\frac{1}{2l^2}|x_i - x_j|^2\right) \tag{C.3.2}$$

with the variance given by σ_f^2 and the length scale of the correlation strength is given by l. These two parameters control the smoothness of the functions generated by the Gaussian process. The squared-exponential kenel is a stationary covariance function as the kernel only depends on the distance between the inputs.

C.3.2 Hyperparameter optimisation

The covariance functions usually have one or more *hyperparameters*; e.g. the squared-exponential kernel discussed before has two (σ_f^2 and l). These parameters are called *hyper*parameters as they are parameters of a non-parametric model. Using training data these hyperparameters can be optimised.

This training can be performed by computing the probability of the data given the hyperparameters θ using the *log marginal likelihood* (or sometimes called the evidence):

$$\log p(\mathbf{y}|\mathbf{X}, \theta) = -\frac{1}{2}\mathbf{y}^T\mathbf{\Sigma}^{-1}\mathbf{y} - \frac{1}{2}\log\det(\mathbf{\Sigma}) - \frac{n}{2}\log 2\pi \tag{C.3.3}$$

Note that we assume a zero mean GP in above definition of the likelihood. By maximising the marginal likelihood we can find the most optimal hyperparameters given the data; Figure C.1.1 shows an example of the effect of the hyperparameter optimisation.

Appendix D

Localisation using Gaussian Process Latent Variable Models

At the start of this indoor localisation research we evaluated several techniques. A promising technique is that of Ferris, Fox, and Lawrence (2007) who proposed WiFi-SLAM: an algorithm which uses *Gaussian Process Latent Variable Models (GP-LVM)* for indoor localisation of users by creating a mapping between RSSI measurements and locations. This technique has a few important merits:

1. No motion data or other user-specific information is required. This removes the need for sensors measuring user motion.
2. The only input data consists of RSSI measurements from different points in time measured at the user's location. This requires that a users carries a device that can measure RSSI.
3. No site surveys or other environment information is required beforehand.
4. No model mapping RSSI to distance is need. The algorithm works directly on the raw RSSI data.

We evaluated the GP-LVM approach using a pilot study as, besides the points above, it shows promising results in terms of performance. In this chapter we will briefly describe the workings of GP-LVM and how it can be applied to indoor localisation. Furthermore, we will explain some of the changes we considered to improve performance. We conclude with the results from our pilot which consisted of a small simulated experiment to show the application of the technique. Eventually we chose not to use the method as the merits did not outweigh the drawbacks: The GP-LVM approach is an offline method and the work by Ferris et al. (2007) focussed only on locating users. Extending the method to also support locating devices (such as the work by Hollinger et al. (2011)) requires additional input and sensors.

For understanding of GP-LVM a basic understanding of probability distributions and *Gaussian Processes* is a necessity; a short tutorial on this can be found in appendices B and C.

The reader that is primarily interested in the results of our pilot, and the rationale behind not choosing the GP-LVM method, can skip the explanation of the GP-LVM technique and continue to sections D.3 and D.4.

D.1 Gaussian Process Latent Variable Models (GP-LVM)

Gaussian processes can be used for supervised learning. Using labeled data (i.e. the values of our input $\mathbf{X}$ are known) we can predict new unseen data points. In GP-LVM we are not interested in predicting new data points but more in finding structures in the data; a form of unsupervised learning. The general assumption is that some high dimensional observed data, $\mathbf{Z}$, may be represented or originate from a lower embedded (or unobserved) space, $\mathbf{X}$, and that we are interested in modelling this lower dimensional space.

Models that utilise this approach are called *latent variable models (LVM)*. In general, a latent variable model relates a set of observed variables, $\mathbf{Z} \in \mathbb{R}^{N \times d}$, to a set of latent (or hidden) variables, $\mathbf{X} \in \mathbb{R}^{N \times q}$, using a set of parameters $\mathbf{W} \in \mathbb{R}^{d \times q}$ (with d, q denoting the amount of features in the two spaces, $d > q$ and N the amount of samples) (Lawrence, 2004, 2005). Examples of latent variables are psychological traits such as extraversion or conscientiousness; these cannot be directly measured but can be inferred using measurable variables. Another example, applicable here, are physical locations (in a 3D space) with a mapping to observed signal strength measurements. A simple linear LVM will assume the following relationship:

$$\mathbf{z}_i = \mathbf{W}\mathbf{x}_i + \epsilon \tag{D.1.1}$$

with ϵ modelling noise of the observed variable. LVM's assume that, when controlled for the latent variable, the observed variables are independent of each other. In general (regardless of linearity) the relation can be defined as through a parameterised function[1]:

$$z_{ij} = f(\mathbf{x}_i, \mathbf{w}_j) + \epsilon \tag{D.1.2}$$

with z_{ij} being a singular observation, $i \in [0, N)$ and $j \in [0, d)$.

D.1.1 Probabilistic principal component analysis

A well known dimensionality reduction algorithm is *principle component analysis (PCA)* which tries to find a set of principle components (which size is smaller than the amount of features) that captures the most variance of the data. More precisely, the first principle component accounts for the largest amount of variance in the data, and each succeeding one for the highest variance given that it is orthogonal to the preceding components. The largest drawback of this method is that we cannot write down a likelihood of the data given the model. The absence of a probabilistic model is addressed in an extension of PCA: *Probabilistic principal component analysis (PPCA)* (Tipping and Bishop, 1999).

D.1.2 Non-linear LVM

The LVM is defined probabilistic and the latent variables can be marginalised out of the model. For the marginalisation we require a prior distribution on the latent variables $\mathbf{X}$, usually a standard normal distribution is chosen for this. The parameters $\mathbf{W}$ can then be found by maximising the likelihood with respect to $\mathbf{W}$ (like the parameters in the optimisation of general GP's). See Lawrence (2005) for an overview of this approach.

[1]Note that the parameterised function is equal to Equation C.1.1 from Appendix C, but here the parameters of the model are explicit (through $\mathbf{w}_j$).

When the mapping from a point $\mathbf{z}$ in $\mathbf{Z}$ to a point in $\mathbf{X}$ is linear and $\mathbf{z}$ has a Gaussian distribution, then the mean and covariance function can be computed easily; if not, approximation is required.

However, in many cases the mapping from the observed to the latent space is non-linear. Due to this non-linear mapping we lose the prior on the latent space and can no longer marginalise $\mathbf{X}$. I.e. the real challenge is to propagate the prior probability distribution through the non-linear mapping.

Gaussian process latent variable models (GP-LVM) (Lawrence, 2004) can be seen as a non-linear probabilistic extension of PCA and address this by treating $\mathbf{W}$ as latent variables instead of maximising their likelihood. It is however intractable to marginalise both $\mathbf{W}$ and $\mathbf{X}$ (in terms of computational complexity). Thus, without resorting to approximation, we need to choose one of these sets to marginalise. In GP-LVM the parameters are marginalised and the latent variables ($\mathbf{X}$) are optimised.

To view the parameters as latent variables we define them as random variables and define a prior:

$$p(w_{ij}) = \mathcal{N}(w_{ij}|0,1) \tag{D.1.3}$$

$$p(\mathbf{W}) = \prod_{ij} p(w_{ij}) \tag{D.1.4}$$

It has been shown that given this Gaussian prior, maximising the likelihood with respect to the latent variables $\mathbf{X}$ still leads to principle component analysis. The marginalised likelihood then becomes:

$$p(\mathbf{Z}|\mathbf{X},\epsilon) = \prod_{j} p(\mathbf{z}_j|\mathbf{X},\epsilon) \tag{D.1.5}$$

$$p(\mathbf{z}_j|\mathbf{X},\epsilon) = \mathcal{N}(\mathbf{y}_j|\mathbf{0},\mathbf{X}\mathbf{X}^T + \sigma_n^2\mathbf{I}) \tag{D.1.6}$$

$$p(\mathbf{Z}|\mathbf{X},\epsilon) = \prod_{j} \mathcal{N}(\mathbf{y}_j|\mathbf{0},\mathbf{X}\mathbf{X}^T + \sigma_n^2\mathbf{I}) \tag{D.1.7}$$

Note that here $\mathbf{z}_j$ is a vector as we marginalised the parameters. By maximising the likelihood we can determine the values of $\mathbf{X}$ that have the highest likelihood given our observations $\mathbf{Z}$.

Although we assumed the parameters to be latent, the covariance function in Equation D.1.7 is still a linear function (namely in the form of the covariance matrix $\mathbf{X}\mathbf{X}^T + \sigma_n^2\mathbf{I}$). When we replace this function by a function that allows non-linearity we obtain a non-linear LVM. These non-linear LVM's are however more difficult to optimise; gradient based optimisation methods are often used for this.

D.2 GP-LVM for localisation

Now that we've briefly introduced the GP-LVM we can utilise this method for the localisation of users in a building. Given a space with n wireless devices. Each device i has a position vector $\mathbf{s}_i \in \Re^d$ with $d = 2$ for 2D environments. All positions of these devices can be combined in a position matrix $\mathbf{S} \in \Re^{n\times d}$. The location of a user traversing through the space at time t can be described with $\mathbf{u}_t \in \Re^d$. The aggregated matrix of the user's movement pattern is $\mathbf{U} \in \Re^{m \times d}$ with m denoting the amount of observations. Both $\mathbf{S}$ and $\mathbf{U}$ are latent variables and are independent of each other (i.e. given the measurements the position of a user does not influence the position of a device and vice versa).

At time t the user receives signal strength measurements of all nodes within range, described by $y_{ti} \in \Re$ for a node i. We assume devices broadcast a unique id or can be otherwise identified. Our observation matrix is then $\mathbf{Y} \in \Re^{m \times n}$. The relationship between the signal strength and the positions can be described using a parameterised function:

$$y_{ti} = f(\mathbf{s}_i, \mathbf{u}_t, w) + \epsilon \tag{D.2.1}$$

Noise is modelled by ϵ and the parameters are made explicit through w. This function implies a probabilistic relation between the signal strength, the location of a device and the current location of the user:

$$p(y_{ti}|\mathbf{s}_i, \mathbf{u}_t, w) \tag{D.2.2}$$

As $\mathbf{Y}$ is dependent on $\mathbf{S}$ and $\mathbf{U}$ the likelihood of our model becomes:

$$p(\mathbf{S}, \mathbf{U}, \mathbf{Y}) = p(\mathbf{Y}|\mathbf{S}, \mathbf{U})p(\mathbf{S})p(\mathbf{U}) \tag{D.2.3}$$

However, using this model it can be difficult to derive the location matrices $\mathbf{S}$ and $\mathbf{U}$ as we have two latent variables. Therefore we combine $\mathbf{S}$ and $\mathbf{U}$ into a single latent variable $\mathbf{X}^{m \times n+1}$ where each $\mathbf{x}_{i,j} \in \Re^d$ represents the location of a device or human j at tilmestep i. Our model then reduces to:

$$p(\mathbf{X}, \mathbf{Y}) = p(\mathbf{Y}|\mathbf{X})p(\mathbf{X}) \tag{D.2.4}$$

D.2.1 Dynamics model

We constrain our latent space $\mathbf{X}$ by extending the GP-LVM with a dynamics model (Wang et al., 2006). We utilise a similar approach as Ferris et al. (2007) but extend this to account for the devices; we model this through a hyperparameter β_i which is 1 if i is a device and zero otherwise. The dynamics are modelled by the product of independent constraints:

$$p(X) = p(\text{distance})p(\text{orientation}) \tag{D.2.5}$$

We constrain the walking speed of users and constrain the locations of devices to zero using an Gaussian prior:

$$p(\text{distance}) = \prod_{t=0}^{m-2} \prod_{i=0}^{n} \begin{cases} \mathcal{N}(dist|0, \sigma_{\text{node}}) & \beta_i = 1 \\ \mathcal{N}(dist|\Delta_t \mu_v, \Delta_t \sigma_v) & otherwise \end{cases} \tag{D.2.6}$$

$$dist = ||\mathbf{x}_{i,t+1} - \mathbf{x}_{i,t}|| \tag{D.2.7}$$

Here μ_v and σ_v are the parameters of the walking speed model, Δ_t the time difference between the two measurements and σ_{node} noise regarding the locations of the devices. Devices are constraint to zero-movement.

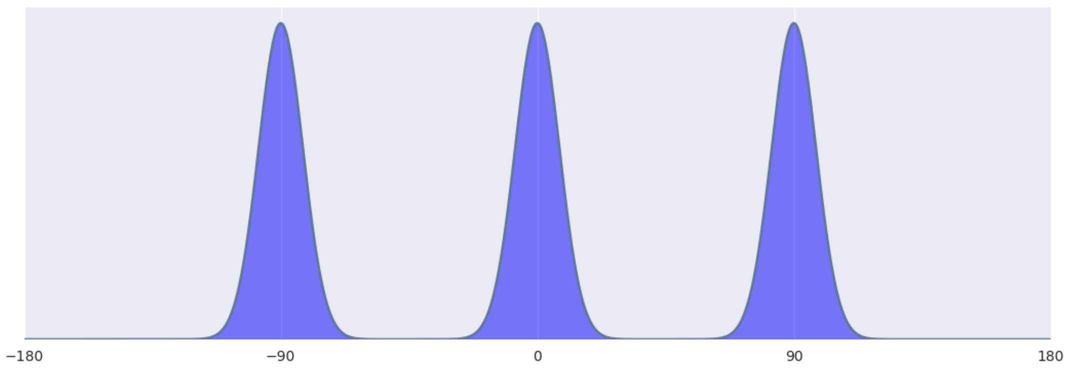


Figure D.2.1: Multi-modal distribution on the orientation between steps. In our model we favour straight paths and turns.

A restriction on orientation between two consecutive points is modelled by a multi-modal distribution (Figure D.2.1). Where Ferris et al. use a zero-mean Gaussian we argue that this model is to simple. We do however assume that users of the building will generally walk in straight lines and that directional changes can be roughly modelled by 90° turns.

$$p(\text{orientation}) = \prod_{t=1}^{m-2} \prod_{i=0}^{n} \begin{cases} 1 & \beta_i = 1 \\ \sum_{\omega}^{\Omega} w\mathcal{N}(\theta_{i,t}|\omega, \sigma) & \textit{otherwise} \end{cases} \tag{D.2.8}$$

Here $\theta_{i,t}$ is the orientation between two points ($t-1$ and $t+1$) on the path of the user, $\Omega = \{-90, 0, 90\}$ and $w = \frac{1}{|\Omega|}$. The orientation constraint is only applied to the trace of the user, not to the devices as these have fixed positions.

D.2.2 Local distance preservation

We constrained our model so that similar locations have similar signal strengths; this is assured by the characteristics of the GP-LVM. The other way around, similar signal strengths mean similar locations, is also valid if the density of devices is high. Only with a high density we can assume that signal strength fingerprints will be roughly unique. If node density is low other measures need to be used (Huang et al., 2011). In our WSN setup we assume that the node density is high and therefore our model can benefit from this constraint.

GP-LVM focusses on dissimilarity preserving: points that are distant in the observation (or data) space $\mathbf{Y}$ will be distant in the latent space $\mathbf{X}$. Our signal strength to location constraint is however a local distance preservation: similar signal strength signatures (i.e. close in the observation space) should map to locations that are close. Note that this local distance preservation is not applicable to all data sets; here we have a well defined latent space which is suitable for these constraints.

Local distance preservation is not a standard characteristic of GP-LVM. To ensure a smooth mapping from signal strength to locations we can utilise the LL-GPLVM (Urtasun et al., 2007) which builds upon Locally Linear Embedding (Roweis and Saul, 2000) or *back-constrained* GP-LVM (Lawrence and Quiñonero Candela, 2006). Because our latent space is strictly defined ($\Re^2$ or $\Re^3$) the back-constrained GP-LVM is the most applicable.

D.3 Simulations of user-only localisation

As part of our pilot a small experiment was conducted to test how well the GP-LVM localisation performs. For this we constrained our self to to a version closely related to the WiFi-SLAM approach. In a simulated world devices were placed at random locations. A simulated user walked a pre-defined path and recorded RSSI measurements from the devices. The ground truth of the path and the device locations can be seen in FigureD.3.1.

After the recording both PCA and GP-LVM were used to estimate the user's path from the RSSI measurements. The *GPy* (GPy-authors, 2012–2014) library was used to perform the computations. Different noise levels were implemented to test the performance of the system given measurement noise.

These first results showed that, given only RSSI measurements as input, the GP-LVM method is able to deduce the user's path. Noise does however have a large effect on the performance. The results are shown in Figure D.3.2.

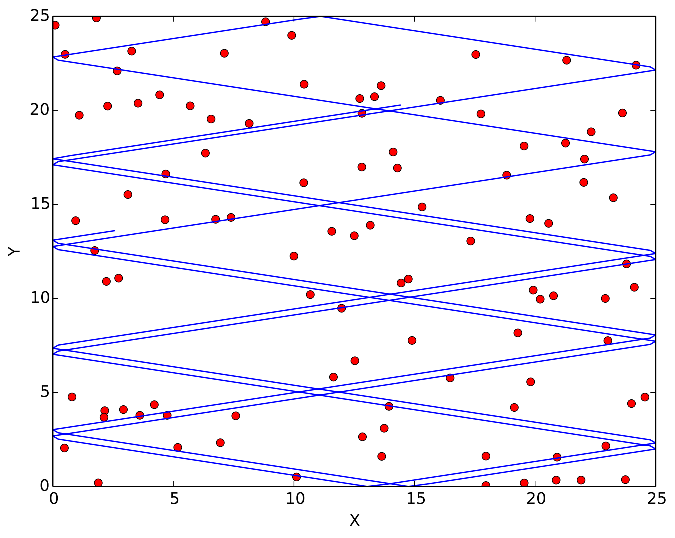


Figure D.3.1: Ground truth of the user's walk pattern (blue lines) and device locations (red dots).

D.4 Problems with GP-LVM localisation

Our simulation showed that GP-LVM is applicable to indoor localisation of users. We however also focus on the localisation of the devices; something the base WiFi-SLAM approach of Ferris et al. (2007) did not address. Our model, in which we incorporated device locations as part of the latent space, is however not able to solve our problem. The problems with the GP-LVM approach are as follows:

Dimensionality: Our representation of our latent space (i.e. the path of the user and the locations of devices), as defined in the previous section, contains a significant flaw: We cannot perform dimensionality *reduction* when our latent space is bigger

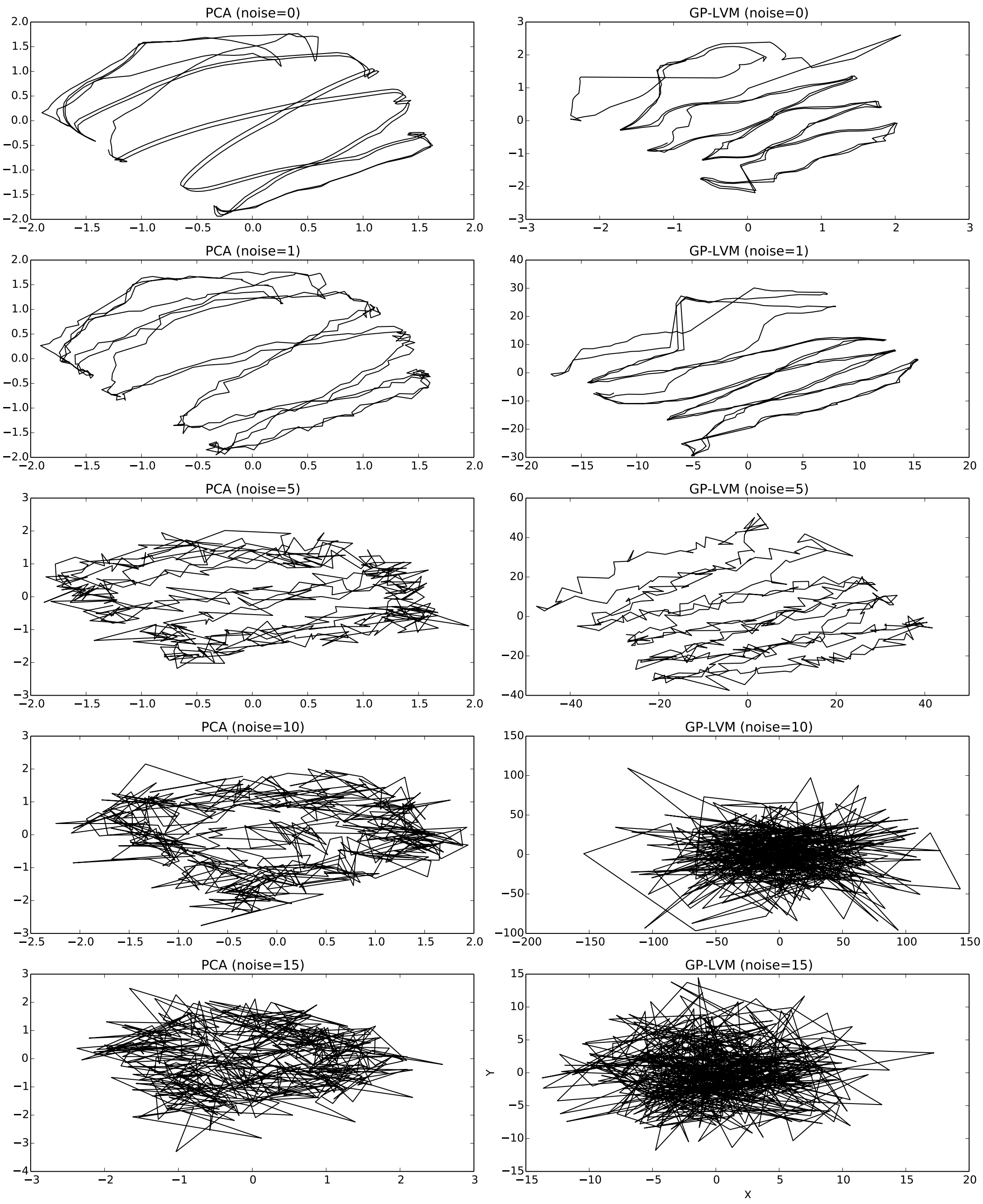


Figure D.3.2: Result of the pilot study. Showing the estimated user path for both the basic PCA algorithm and GP-LVM. When the noise level is increased the localisation error increases drastically.

than our observed space. Our latent space has a dimension of $M \times N + 1$ given N devices and 1 user. The observed space has only a dimension of $M \times N$.

To address this issue we, if we want to apply GP-LVM, will have to introduce additional input to our system to increase the dimensionality of our observation space. This contrasts, however, with our goal to reduce the required additional hardware and prior information.

Offline: The GP-LVM method is an offline method as it uses the full user trace to perform localisation. This is not necessarily a problem for locating devices; their location will change less often than users. For users this is problematic, localisation should start when someone enters a building and this location information should be available instantly. When there is a delay a smart space system cannot react in real time on users' presence.

We weighed the benefits of the GP-LVM method with the problems described above. Eventually we favoured a different approach (the FastSLAM method) and decided to not use GP-LVM as the base of our algorithm.