What if everything is an illusion and nothing exists? In that case, I definitely overpaid for my carpet.

– Woody Allen

University of Alberta

INDOOR LOCALIZATION WITH PASSIVE SENSORS

by

Meisam Vosoughpour Yazdchi

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Spring 2013
Edmonton, Alberta

To my parents
With whom I can hope.

Abstract

In this thesis, a framework is described that is designed to perform indoor localization in the SmartCondo^{TM}. A significant aspect of the framework is that it mainly operates on the basis of binary sensors – including motion sensors and occupancy sensors – and it primarily involves geometric computations. In addition, switch-type sensors have been incorporated. We have specifically designed and implemented a geometry library to facilitate the necessary computations, as well as a simulation tool to simulate the environment and its sensors. Compared to previous related research work, we adopt a more realistic environment model, as well as models for a person's body, and models for the sensors. In the experiments conducted, when the sensors are assumed to behave in an ideal fashion, we have achieved $67cm$ and $49cm$ as mean localization error for minimum coverage and dense coverage sensor configurations respectively. Under a more realistic sensor behavior model the corresponding numbers are $69cm$ and $62cm$ respectively.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

[6] introduces the term "Smart Home" as *"a home that has both a set of sensors to observe the environment and the actions of its occupants, and a set of actuators to automatically control home devices to improve the occupants' experience"*. The SmartCondo™ is an instance of the "Smart Home" agenda, but more specific to healthcare related concerns [6]. For the most part, the SmartCondo™ aims to constantly monitor the location and activities of a patient or an elderly person living alone in a condo. The latest version of the SmartCondo™ infrastructure and implementation consists of three main layers: (1) Sensor network layer, (2) Data storage and processing layer, and (3) Visualization layer. Briefly, in the sensor network layer, the data from all sensors are collected to a collecting "sink" node; the sink node then feeds the data storage layer using the MQTT protocol [1]. In the second layer, all received data are stored and processed. In the third layer, the results from the various types of processing performed at the second layer are visualized and analyzed. Analyses and visualizations are considered for various groups of end-users, notably including clinicians [6]. In this structure, any kind of processes which are demanded by the end-users can be added in the second layer. Currently, the most important requirement is the updated location of the person staying in the SmartCondo™. From the location information, clinicians can assess the degree of activity of the patient, e.g., visits to the washroom or time spent preparing food or time spent lying in bed. Apart from assessing the overall level of energy of a patient, the clinicians can gain valuable insights from relating activity with the medication protocol followed by the patient and patients physical condition. This is because, even coarse-grained location of the patient can be used as a proxy for his/her activity while, additionally, the temporal aspect of the location (e.g., how much time spent in the kitchen area preparing food) can also be an indicator of the subject's health disposition. Hence, there are ample reasons to consider that the most important processing performed in the second layer is the task of indoor localization which is precisely the topic of this thesis. Towards this end, we have studied previous methods of indoor localization and proposed a different indoor localization algorithm which better suits the specific application.

The problem of indoor localization has been extensively researched over the past decade. Natu-

[1] Information about MQTT is available at http://mqtt.org

rally, the problem most similar to the indoor localization is the outdoor localization. Today, outdoor localization is considered an essentially solved problem due to the use of the Global Positioning System (GPS). However, GPS is not suitable technology indoors because the satellite transmissions are usually impossible to receive due to walls and other obstructions [19]; hence, the problem of indoor localization requires a different approach. The investment on solving indoor localization is not surprising due to its various real-world applications: (1) medical and health-care applications such as indoor navigation systems for the blind [15][23], constant measurement of a person's vital signs in a room [20][27][16], or proof-of-concept environments such as the SmartCondo^{TM} [6] which is aimed to monitor and locate patients or elderly living alone in a condo, (2) inferencing applications such as activity recognition of children at different ages [42], (3) security applications such as monitoring and tracing people in highly secure and strategic buildings [25], (4) everyday applications such as extending GPS-based localization once a user enters a mall or shopping center [3], etc.

There are literally *many* reported methods for indoor localization. In a recent survey, [35], approximately 20 different technologies used for indoor localization are reported and are classified by the type of sensors used and the general technique they employ. The same survey, provides a long list of relevant technologies: contact sensors, pressure sensors, chemo sensors, photo detectors, cameras, thermal images, break beam sensors, scalar range-finders, scanning range-finders, tomographic sensors, EF sensors, Doppler-shift sensors, motion sensors, seismic and inertial sensors, microphones, wearable environment recognition, wearable SS device-to-device rangers, wearable TOA/TDOA device-to-device rangers, etc. It is interesting to note that the listed technologies can be used in a variety of ways and, hence, each of them has led to the production of various research papers demonstrating different solutions to the localization problem. Based on whether the localization method requires the target to wear a device or not, indoor localization methods are classified into two categories: "wearable" and "non-wearable" (the survey in [35] calls them "instrumented" and "uninstrumented" systems). Additionally, major wearable methods sub-categories are (a) methods based on "*Dead Reckoning*" [9] [12] [36] [28], (b) methods based on properties of wireless packet transmission between a mobile device and some stationary nodes such as TOA/TDOA [43] [2] [13], and (c) methods based on the perceived pattern of radio signals such as WiFi and GSM [31] [39]. Non-wearable methods also use a variety of technologies: regular (visible light) cameras, thermal imagers, passive infrared sensors, contact sensors, pressure sensors, etc [35].

Despite the fact that some of the previously proposed methods have been reported to achieve remarkably higher values of accuracy than some others, we cannot necessarily call the method with the highest accuracy a "universally best" indoor localization method. Specifically, while the goal of all these methods is the same, i.e. indoor localization with the minimum error, there are several application-dependent criteria to decide if a specific indoor localization method is feasible (or infeasible) and acceptable (or unacceptable). Some of the factors that come into the picture in order to determine feasibility and acceptability are: equipment and deployment cost, number of individu-

als/objects localized, desirable precision, user acceptance, and the deployment environment. These are explained with examples in the next chapter.

Our indoor localization system is an uninstrumented method; because of our application we insisted not to use an instrumented technology. In SmartCondo™ [6], it is often the case that a single elderly person or a patient is going to be localized, and it is strongly preferable for them not to wear any devices. On the other hand, we wanted to have a localization system that could be easily and quickly installed in a new personal condo or a new room inside a hospital. The privacy of people did matter as well, meaning that we could not use cameras. For all the above reasons, in the beginning we decided to use passive infrared motion sensors. Our system evolved and now we have used several types of sensors: (1) passive infrared motion sensors, (2) occupancy sensors for beds and chairs with fixed locations, and (3) switch-type sensors. By (3) we mean any sensor which has on/off or open/close states that can be changed by the person, such as magnetic reed switches and electrical current sensors. (1) and (2) can be considered as general cases of binary sensors or *abstract* binary sensors [35]. The design of our localization framework is heavily influenced by the use of binary sensors as the primary available devices. At certain points, we treat motion sensors and occupancy sensors differently, while in other points we just view them as abstract binary sensors. In addition to the main product of this thesis, our work also resulted in two side artifacts: (a) a geometry-library which is designed to make the localization algorithms easy to use and fast (in practice, the processing time for every location update in our framework - using Windows® 7 OS and Intel® Core™ i7-2860QM CPU at 2.50 GHz - is less than 20 milliseconds in our framework); this library will be discussed in chapter 3, and (b) a simulator which simulates the environment, the sensors, the noise, and the movement of individuals to provide the localizer with sensor readings as it would receive from real sensors.

Next, we enumerate our algorithmic contributions in relation to three papers on localization with binary sensors (Kim *et. al.* [18], Shrivastava *et. al.* [34], and Wang *et. al.* [41]). There are certain aspects that all these papers share in common and are improved by our method:

- For simplicity, all the mentioned papers consider the sensing range of each sensor to be circular. In our method, it can be any shape according to the real sensor's specifications. Due to this, in case of real passive motion sensors, all of them have to assume that all sensors are positioned vertically facing down so that the borders of the sensor's sensing range on the floor form the shape of a circle. In fact, by default they assume a two-dimensional field with several sensors each of which has a two-dimensional point as its center and a circle as its sensing range.

- They actually assume that the target is moving in an *infinite* two-dimensional field, with no wall that can potentially block a sensor's range. In comparison, we precisely take into account walls and other obstacles which can partially block a sensor's range. We even consider that a sensor's rays in one room can penetrate the other rooms through open doors.

- The geometry part of all of them is based on the intersection and subtraction of circles, but they do not necessarily describe the relevant algorithmic details. Most of them appear to be performing all the necessary geometric computations at every iteration, while, as it will become clear later, we emphasize the pre-processing step to create an efficient geometric model that will be subsequently used for real-time localization.

- The final output of all mentioned papers is a single point which is a guess for the last known location of the individual. But, in addition to this point, we also output an area of confidence along each estimate. By exploiting the short-term history of our computed person's positions, we prune and narrow down this area of confidence as much as possible.

- All above methods, consider, for simplicity, that a localized person is a single point in space, i.e., without any "thickness" whereas we extend the model to considering each individual as a cylinder, thus approximating the 3-dimensional nature of the individuals.

- With respect to motion sensors, they simply assume that a person is always moving or, in other words, even a stationary object in the sensing range triggers a sensor. We have taken into account the fact that the person may be stationary sometimes, in which case sensors may not be triggered even if the person is inside their sensing range.

- Finally, they do not consider the nature of the sensor signals in the real world. For instance, even when a motion sensor is triggered, its signal is not constant but it may be oscillating, and this oscillation is not necessarily regular. We include simple algorithms to handle this oscillation and obtain a more accurate motion detection signal.

In addition to binary sensors, we have also used switch-type sensors. The successful integration of data from switch-type and binary sensors is an important achievement of our framework. We have run experiments in a 3-room $\approx 120m^2$ house. We have achieved a mean error of $\approx 69cm$ using a minimum-coverage configuration of non-ideal binary sensors. This mean error improves to $\approx 62cm$ when we increase the number of sensors by 50%. By integrating the signals from nine switch-type sensors, an average improvement of $\approx 16\%$ is further obtained.

1.1 Overview

In chapter 2, important characteristics of the application of indoor localization are introduced with examples. Then, inspired by the classification of indoor localization methods in [35], we roughly categorize such methods. In the most important section of this chapter, the most relevant sub-category of methods to our method – methods using passive binary sensors – is introduced in more detail. In chapter 3, our geometry library is explained. This library is an important piece of the infrastructure needed to support our algorithms.

In chapter 4, first the general specifications of our localization system is described. Then we explain how we model different entities and concepts in the environment. Continuing, our simulator component is introduced. Next, our framework using ideal and noise-free sensors is elaborated upon. Finally, five main challenges in the real-world employment are enumerated and our solutions to overcome each of them are described.

In chapter 5, five main dimensions of variability for our localization system are introduced to setup appropriate experiments. Then, according to the results of experiments, the effect of each dimension on the accuracy of our system is investigated. Finally in the last chapter, we conclude the thesis and express the most interesting future directions.

Chapter 2

Background

A plethora of indoor localization methods, based on various technologies, have been proposed over the last decade. The desirable qualities of an indoor localization method are application-dependent. Knowing the details of an application and its crucial characteristics, the question becomes: "Which localization method is the best match for the particular application?". As outlined in the previous chapter, we consider five facets of an application's requirements. Here, we illustrate the five facets we consider:

- **Cost:** Cost is playing an important role in determining the best method for indoor localization and its accuracy. For example, in the case of binary sensors, the deployment density of sensors can range from being just enough to provide minimum coverage (with a low expected accuracy) to a super dense sensor placement resulting in a very precise localization. In such a case, if high accuracy is demanded, the number of required binary sensors might become so high that another localization method using a different technology could accomplish the same accuracy at much lower cost. Cost usually includes both the device cost as well as the deployment (man-hours required) cost of the devices and/or their calibration.

- **Number of People:** There are two determining factors related to this criterion: (1) the number of people which are supposed to be simultaneously localized and (2) the total population in the environment who can act as "disturbers." Based on the two characteristics, it might be impossible to localize using a particular technology, whereas another technology can do so. For example, if the number of persons to be localized is more than one, almost none of uninstrumented methods (i.e. methods which do not require the localized person(s) to wear any devices) are capable of performing accurate localization.

- **Desirable Precision:** Every application of indoor localization requires a particular precision; e.g. for guiding a blind person, an accuracy of less than half a meter is required, while in the SmartCondo^{TM} project [6] this precision is just too much and 1.5 meters is sufficient. Of course, once there is a method which accomplishes the "good enough" precision for an

application, employing another method at a higher cost to acquire a better precision would just be a waste of money.

- **User Acceptance:** This characteristic is very similar to the "user-friendliness" for software products. For instance, people are not normally comfortable to be monitored by cameras in more private areas. Therefore, indoor localization inside a house using a method which uses cameras is not acceptable by tenants. Another contributing factor to the user acceptance is the setup complexity of a method. For example, if there is no infrastructure for installing pressure sensors on the floors of an ordinary house, all methods based on such sensors will be ruled out. Another factor is whether the method requires the person to wear a device or not. Localization methods requiring wearable devices are in general less accepted by users, though it is not always true; this factor will be discussed in more detail in the rest of this chapter.

- **Environment:** Based on the type of the building there may exist restrictions on using specific devices, e.g., because of the inability to power devices if they are placed far from a mains plug. The size of the building also matters; the optimal indoor localization solution for a mall is certainly different from that of a condo. Furthermore, for different building layouts, different localization methods are needed; the optimal solution for an open space with very few walls and obstacles is not necessarily the same for a building with lots of walls, small rooms, and corridors; also most methods assume that the building is flat, while there are special methods suited for multi-storied buildings. It is also possible that, in more varied buildings, hybrid localization technologies could co-exist. For example, in narrow corridors, motion sensors can be readily used since people tend to be non-stationery there, while other technologies (like RFID readers) might be employed at the exit points of the corridors and in other locations in the building.

Because of all above application-dependent criteria, there is no universally best solution for indoor localization. In the absence of the above criteria, all methods would have converged to a single best solution. Based on the technology used as well as the above criteria, indoor localization methods are categorized into various classes. The first thing that branches all the methods into two coarse categories is the way the final indoor localization product is going to be used. In continuation of what we mentioned before for the "User Acceptance" criterion, based on weather the target person has to wear a device or not, indoor localization methods are classified into two categories: "wearable" and "non-wearable". There are three groups of localization applications regarding those two categories:

1. The first group *has to* employ an uninstrumented localization method. In other words, all instrumented methods are infeasible/unacceptable for this group of applications. This usually happens when the target person is unaware of the existence of a localization system and that they are monitored. For example, for localization of people entering a highly secure building

at night, it is meaningless to ask them to wear a device before entering. In this case, they might not even be aware they are monitored.

2. The second group *has to* employ an instrumented localization method. This usually happens when the target person is not going to be monitored and the localization data is not going to be used by a third party, but the "end-user" is the target person himself. A tangible example is the well-known outdoor localization system via GPS. It is usually the person who is using the GPS data that acquires them, and hence they have to carry a capable device. The same holds for indoor navigation systems for blind people. In all such applications the wearable device has a double functionality: (a) a device aiding the localization system, and (b) a localization data receiver and a visualization/feedback device to inform the target person being the end-user.

3. The third group of applications have the option to employ both instrumented and uninstrumented methods. But *almost all* of them prefer uninstrumented methods. This usually happens when the target person is not the end-user and a third party is monitoring them, while at the same time they are aware of being monitored, so they may cooperate. A good portion of indoor localization applications belong to this group. For example, in the SmartCondo™ project [6] the patient can be localized by an uninstrumented method or can wear a device if required by an instrumented method; but, obviously an uninstrumented method is much more preferable as long as the minimum accuracy is obtained.

In the rest of this chapter, major technologies of both instrumented and uninstrumented types are briefly introduced and those methods based on uninstrumented binary sensors – which are the most relevant to our localization method – are explained in more detail.

2.1 Wearable (instrumented) methods

Our localization method is an uninstrumented method, so there is not really that much similarity between instrumented methods mentioned here and ours. However, for the sake of completeness, this section briefly introduces the most important instrumented methods according to [35]. In [35], instrumented indoor localization methods are categorized into three main groups:

- One group of localization methods use an approach which is called "*Dead Reckoning*". They assume that a person's new position can be computed by summing the displacement, given a particular movement vector, with the previously known position. These methods use various devices to measure the velocity and/or the acceleration of the person. By these pieces of information as well as knowing the person's position at some previous point in time, they produce position estimates at future time points. Due to this cumulative approach, the localization error may also accumulate, and grow with the distance traveled [35]. Hence, it is common to report error as a function of distance. There are many methods using this approach. They

have evolved over years and their errors range from 2% to, as low as, 0.2% of the distance traveled [35] [9] [12] [36] [28].

- Another group of instrumented methods are based on properties of packet transmission between a mobile device and some stationary nodes. They include methods that use the time of arrival (TOA)[13] [8] of packets to different stationary nodes and compute the distances of the mobile node to the stationary nodes. The mobile node's location is then computed out of those distances via e.g., multilateration. Time difference of arrival (TDOA) [43] [2] and angle of arrival (AOA) [7] [21] are two other techniques used for the same purpose. Some methods in this group can provide under 0.5 meters of localization error.

- The third group of instrumented methods use the pattern of radio signals to localize the mobile node. WiFi [39] [22] and GSM [31] [38] are two kinds of radio signals which are commonly used for this purpose. Their common characteristic is that they are based on inferring the location based on a "map" of known signal vectors collected at known locations.

2.2 Non-wearable(uninstrumented) methods

In this section, the most important uninstrumented methods are introduced; they are roughly classified based on the technologies they employ: (visible light) cameras, thermal imagers, passive infrared sensors, contact sensors, pressure sensors, etc.

Cameras are almost an outlier in this field compared to other types of sensors. While they are relatively cheap and easily found, they provide high resolution and high quality images of the area with a lot of information in them. In one sense, one can think about them as millions of very tiny color sensors covering an area corresponding to the camera view. As cameras provide complex and informative data, extracting information and building the final indoor localization system out of them is very challenging as well. Image processing techniques have contributed to the relevant challenges. The biggest challenge is to person(s) in an image; followed (or complemented) by computing the distance of person(s) to the camera, and eventually estimating the location of the person(s) in the viewed space. Three major techniques are employed in image processing to detect persons in images [35]:

1. Background subtraction is the most common technique, and it is based on the fact that the background of an image corresponds to stationary objects in the environment and, hence, by subtracting the current image from the background image, one can isolate moving objects, and specifically human forms [1] [17]. As long as only a single person is moving in the environment, this algorithm is perfectly suitable, but in environments of frequent and non-smooth movement of other objects, the algorithms could produce errors. To overcome this problem, one good solution is using multiple cameras and utilize a system of stereo imaging [14].

2. Image pattern recognition is another well-known technique in image processing to detect a person in an image. In this technique, numerous sample images of the target person are fed into a classifier and a pattern of the target person's image is trained using machine learning techniques. This trained classifiers are later used in real-time to detect the person in the image [37] [40] [11].

3. Thermal imaging is a relatively newer image processing technique for human detection. The main advantage of this method over "background subtraction" is that it is not sensitive to most of the movement of objects in the background, because the body temperature of the target human is the actual distinguishing factor. By using stereo thermal imagers, high accuracy of human detection is obtained [5]. The main problem with this method is its relatively high cost.

Pressure sensors (e.g. in the form of touch sensitive floor mats) are a tested technology for uninstrumented indoor localization [35][32]. One big problem with this technology is its difficulty of installation. Additionally, the localization algorithm itself is non-trivial than it seems at first glance. If there is only one person in the environment and the person has only one contact point with the floor, both the localization and calculation of the speed would be trivial. In case of a single person, with possibly one or two contact points with the floor (corresponding to the person's two feet), the localization is still not that difficult, but the speed calculation could be complicated. Finally, the most complicated example is one where there are more than one person in the environment each of whom has one or two contact points with the floor at a moment. In this state, different algorithms are proposed to map the contact points to the persons' feet; such as looking at the short history of the movements and assuming that people tend to move in a smooth path, or taking into account the exact amount of pressure by force measurement based on this observation that this force is different for different people [24][30].

2.3 Binary sensors

Ideal binary sensors are a group of sensors which output 1 if and only if the target person in inside the sensor's sensing range; otherwise, they output 0. According to this definition, it does not matter if binary sensors are working in an instrumented or uninstrumented framework; but in this section we specifically mean binary sensors which work in uninstrumented frameworks. In the real world, there is no 100% ideal binary sensor, but, we define a "binary sensor" as an "ideal binary sensor" which is allowed to have an acceptably low rate of error, i.e. it could output 1 when the target person is not in the sensing area (false positive) and vice versa (false negative). Examples of binary sensors are: passive infrared sensors (PIRs), binary pressure sensors, contact sensors, etc. However, in the most comprehensive works in this area, the algorithms are not limited to any particular binary sensor technology; in fact, most methods are working for any type of binary sensors as long as

they meet the general definition of a binary sensor. Therefore, in the following mentioned papers in this subsection, unless explicitly mentioned, all methods by default consider the general model of a binary sensor without limiting it to any specific technology. In the following three papers [18] [34] [41] all binary sensors have circular ranges of radius R, with the center of the sensor being the center of the circle.

In [18] , Kim *et. al.* first employ the traditional *Centroid* method which determines the target's position by the centroid of all sensor centers which have detected the presence of the target. They then introduce their weighted centroid algorithm which assigns a weight to each sensor's center. They compute this weight mainly based on the distance of the target to the sensor's center. They explain an algorithm to compute the target's distance to each sensor at a given time. This algorithm works as follows: for a given sensor in $[t_1, t_2]$ period of time which is the detection time range of that sensor, a constant speed on a straight line for the target is assumed. So, the length of the segment by which the target's trajectory has intersected the sensor's circle of radius R is easily computed as $(t_2 - t_1) * s$, where s is the assumed constant speed of the target. Using the length of the intersecting segment and also knowing R, the average distance of the target to the sensor during $[t_1, t_2]$ is computed. This procedure is performed on all sensors and separate weights are assigned to different sensors based on their distances to the target at a given moment. The final estimate of the target's position will be computed as the weighted centroid of those sensors which have detected the presence of the target at a given time. It is important to be noted about this paper that, they consider all sensors which have 1 signal at a given time, but they do not take the 0 signals of other sensors into account which could have a lot of information, and is usually called the *exclusion information*.

Shrivastava *et. al.* [34] consider a sequence of time stamps : $\{t_0, t_1, t_2, ..., t_{n-1}\}$ where t_i represents the i-th time that the target entered or exited a sensor's circle, i.e. the i-th time that the sensors' signal array changed in one bit. They compute an arc for each t_i, let's call it arc_i. They do not go into details how they compute it, but it appears that they adopt several assumptions to simplify this process. Finally, determine the best trajectory, they try to cross all arcs (arc_0 to arc_{n-1}) with the minimum number of segments.

The work of Wang *et. al.* in [41] is very similar to [34], as they also focus on $\{t_0, ..., t_{n-1}\}$ and find proper arcs corresponding to these moments: $\{arc_0, ..., arc_{n-1}\}$. However, the way they pass a trajectory across these arcs is different. They consider n points which are the mid-points of the arcs. Then, they define an appropriate $k < n$, and calculate the best line (in the least-squares sense) from the first k points of the array of n points; then, they proceed forward by one point and calculate a new line for the next set of k points. At any stage, if the slope of the line differs from the slope of the previous stage's line by more than a threshold, a new segment will be constructed and the algorithm continues with the new slope. In general, this technique is very similar to the work in [34].

2.4 Summary

Papers on localization with binary sensors contain at least two possible areas of improvement: First, they tend to lack clarity in explaining details of the used algorithms. This makes it very difficult to replicate a completely functioning version of such systems. For example, when describing geometric computations on circles and arcs, they do not elaborate; so, even some obvious special cases and counter-examples are left unexplained. It is reasonable to assume that they often apply even more simplifying assumptions to avoid corner cases. For example, in the models of [41] and [34] no description is provided as to what happens when one circle is completely contained within another one. In such a case, the border between them is not well-defined, because they do not intersect at two points, while such assumptions of intersection are crucial for their proposed techniques. This case is avoided by further assuming that all circles have the same radius. Similarly, their algorithms are unable to account for obstacles – e.g. pieces of furniture – even if the shape of all obstacles were limited to be circular. They consider many other simplifications, such as adopting a 2D model, avoiding the real 3D nature of the problem, or not taking into account walls and obstacles that may completely block sensing ranges, etc.

Secondly, the mentioned previous work ignores some real-world requirements: (a) *Body model*: Since their space model is 2D, the person's body has no height. In addition, it has no thickness and is simply considered as a single point in the 2D plane. (b) *Physical details of the environment*: They do not model the building at all. So, any detailed information about the building model is obviously lacking, such as the floor-plan of each room, the height and model of the ceiling in different parts, the exact model and location of doors and obstacles, the impact of various types of obstacles on the sensing ranges, etc. (c) *Real model of sensors*: Their model of sensors are abstract to the point of being *too* abstract to be useful. Each sensor is modeled as a 2D point with a 2D circle of a constant radius that represents its sensing range. They neglect the fact that a motion sensor is not triggered when the person is stationary in its sensing range, or that it can produce false positive or false negatives. The real signal pattern of sensors are also ignored, e.g. the oscillating pattern of positive signals in motion sensors, not to mention the corner case of (some) malfunction sensors.

In this thesis, we have attempted to address all the above points in a coherent, and as complete as possible, fashion with the purpose of deploying the algorithms and techniques to actual real-world environments.

Chapter 3

The geometry library

Our localization framework contains several modules which rely heavily on geometric computations. Most of these computations consist of interactions between ordinary geometric objects: points, segments, polygons, and regions. In other words, the better we can design and implement an efficient geometric computation component, the easier we can examine various localization algorithms, expand our localization framework, and consequently improve the accuracy of the localization. Towards this goal, we have designed and implemented a geometry library which supports a dynamic query facility. This facility allows one to query the interaction between two or many geometric objects of three varieties: Point2D, Polygon, and Region. This is supported by a class named Map which will be discussed in the last section of this chapter. In the following subsections, all above-mentioned identities except for Point2D, which has a trivial definition, will be introduced. Since we have used an object-oriented design to implement the geometry library, all five identities are actually classes in our model and the conventions of object-oriented literature will be used to describe them. Methods of classes are presented in C++ style. It should be noted that those methods whose algorithms are not described are all well-known algorithms in computational geometry which we have found in, e.g., textbooks [10][29][4][33].

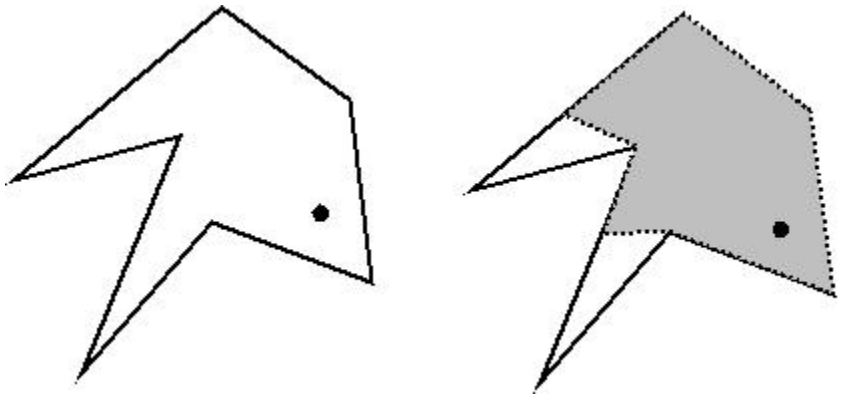

Figure 3.1: The gray area shows the visible area from the source point.

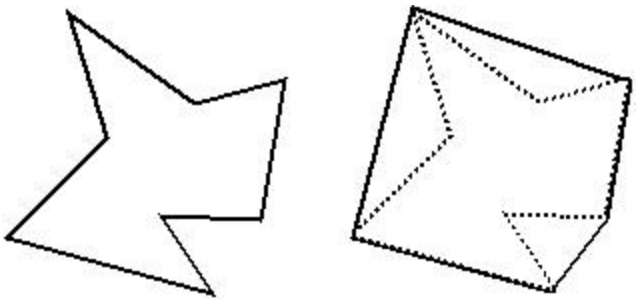

Figure 3.2: The left polygon is the original polygon, which is dotted in the right figure and surrounded by its minimum convex hull polygon.

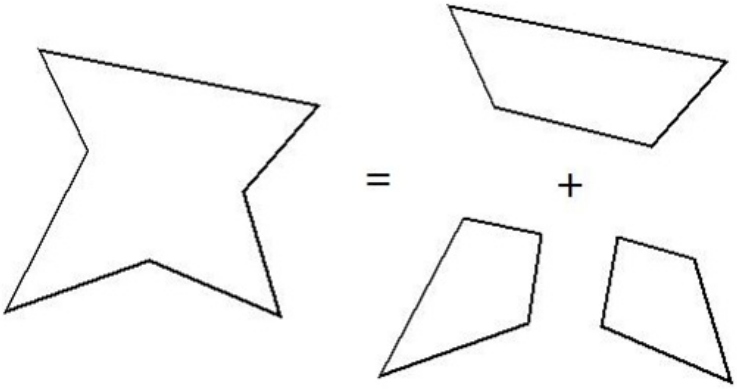

Figure 3.3: The non-convex polygon is partitioned into three convex polygons.

3.1 Polygon

A Polygon object is specified by an array of Point2D objects; every two consecutive points as well as the first and the last points in the array are connected by an edge in the polygon. The important methods of the Polygon class are as follows:

- *bool Polygon::isVisible(Point2D source, Point2D destination)*: Knowing that a light ray travels in a straight line and assuming that edges of a polygon are rigid and do not let the light pass through them, this method checks if two given points are visible from each other.

- *Polygon Polygon::getVisibleArea(Point2D source)*: This method computes the set of all points which are visible from a given source and returns this set of visible points as a Polygon object. Figure 3.1 shows an example of this method's operation.

- *Polygon Polygon::getConvexHull()*: This method returns a Polygon object that represents the minimum convex hull of the current polygon. Figure 3.2 shows an example of this method's functionality.

- *vector<Polygon> Polygon::partitionIntoConvexPolygons()*: This method, divides a polygon into the minimum possible number of convex polygons. Figure 3.3 shows an example of this

method's functionality.

- *Polygon Polygon::expandPolygon(double dist)*: This method expands the polygon by $dist$ units of distance. More precisely, Polygon b is a $dist$-expansion of Polygon a if and only if b includes a, and for all single points named p on edges of b, the minimum distance of p to any point on a is *exactly* $dist$. Two important points should be noted here: (1) If polygon a is not convex, polygon b, with the given definition, does not necessarily exist. This is shown in figure 3.4 by an example. Therefore, we assume this method only operates on convex polygons, otherwise first the minimum convex hull of the polygon is computed and then the expansion is performed. (2) As shown in figure 3.5, the precise expansion of a convex polygon does not result in a sequence of segments to form a new polygon, but a sequence of segments and arcs are generated. Since we want to express the result as a simple polygon, we convert all arcs to segments which results in an acceptable approximation for the purpose of indoor localization.

- *double Polygon::getArea()*: This method computes and returns the polygon's area.

- *Point2D Polygon::getCenterOfMass()*: This method computes the center of mass of the polygon and returns it as a Point2D object. The center of mass of an area is the average of the coordinates of all points in that area.

3.2 Region

The class Region is designed to be able to represent geometric shapes that cannot necessarily be represented by a polygon. A simple example of such figures is a ring-like shape which is shown in figure 3.6. A Region is specified by an object of Polygon called $big\text{–}polygon$ as well as an array of Polygon objects named $holes$. Any 2D connected topology has a surrounding polygon around it; in the Region representation of this topology, $big\text{–}polygon$ refers to this surrounding polygon around the topology. Any 2D connected topology also has a set of internal holes which are all included within $big\text{–}polygon$; these holes are stored in an array of Polygon objects named $holes$ in the Region class. Figure 3.7 shows a 2D topology and explains how it is represented as a Region object. Any 2D connected topology can be represented by a Region object. The important methods

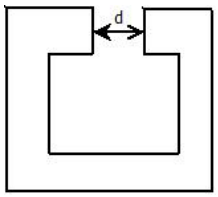

Figure 3.4: If this polygon is expanded by more than d/2 units of distance, the resulting shape will be a collapsed polygon.

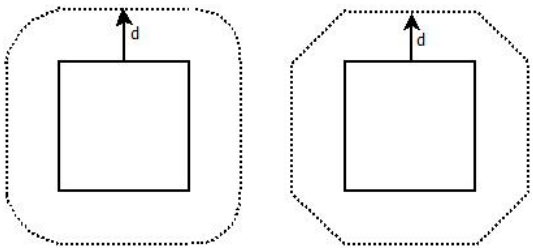

Figure 3.5: In the left figure, the square is accurately expanded which results in a (dotted) sequence of segments and arcs. The right figure shows our polygon approximation.

of the Region class are:

- *double Region::getArea()*: Computes the region's area, which is the area of its $big - polygon$ minus the total area of its holes.
- *Point2D Region::getCenterOfMass()*: Computes the center of mass of the region.
- *Region Region::expandRegion(double dist)*: Logically, in expanding a region, we should expand its big polygon and shrink each hole. We apply an approximation and neglect shrinking the holes.

3.3 Map

The Map class is aimed to support the interaction between instances of polygons and/or regions. The most important application of this class in introduced in the next chapter. We will see that, in the precomputation stage of our localization framework, the sensor ranges are modeled as 2D regions on the floor. The Map class is used to build the geometric model of all the regions which are generated by the intersection of these sensor ranges. Although we briefly explained the most important application of the Map class, this class is still a general geometric tool and does not belong to any specific application.

Conceptually, an object of Map is initially like a *blank page*. The Map class has a method named $addRegion$; by adding any new region to the map, that region is actually drawn on the map page. After all target regions are added to the map, the $preprocessAll$ method can be invoked. After this method is invoked, a model of the entire page is built up and will be used in future queries. In order

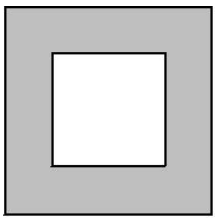

Figure 3.6: A ring-like region; the gray area shows inside the region, and the white square is a hole.

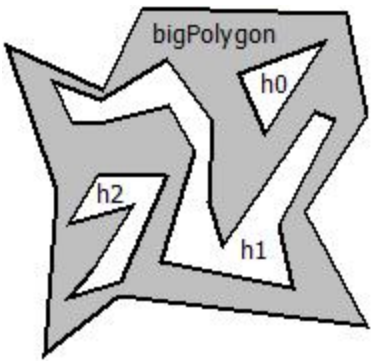

Figure 3.7: A 2D topology represented by a Region object with three holes.

to give a more clear understanding of Map's functionality, we will explain its role in a particular scenario:

- In the first step, once an instance of the Map class called map is defined, it is blank and contains no regions. The method *Map::addRegion(Region)* allows us to add as many polygons and/or regions in map. It should be noted that any polygon should be added as a Region object, as it is simple to see that any polygon is actually a region with no holes. This list of added regions is stored in an array named $originalRegions$.

- Second, after all the regions are added to map, $preprocessAll$ is ready to be invoked. This function operates in two steps. First, it invokes the method *Map::initializePlanarGraph()*. This method models all the regions in the map as a planar graph. To do so, it first initializes an array of points named $eventPoints$. All vertices of all polygons inside each region in $originalRegions$ (including each region's $bigPolygon$ and $holes$) are added to $eventPoints$. Also, the intersection points of any edges of different polygons inside all regions are added to $eventPoints$ too. The array $eventPoints$ contains all nodes of the planar graph, and any pair of nodes which are immediately (with no other node in between) connected by a polygon's edge in map are considered as connected nodes in the planar graph.

- Third, after the planar graph is initialized, in the next step, $preprocessAll$ invokes the method *Map::findRegions()*. It executes an algorithm to determine the faces of a planar graph [26]. After all faces of the planar graph are found, all detected regions are stored in an array of regions called $mapRegions$. It should be clarified that, during this process, two points are considered to be in the same region if and only if they are reachable from each other; and two points are reachable from each other if and only if there is a path from one point to another which does not pass through any edge.

- Finally, an array of CodedRegion objects are generated which is named $codeRegions$ which has the same length as $mapRegions$. A CodedRegion object is specified by a Region object named $region$ and a boolean array of length $\|originalRegions\|$ which is named $code$.

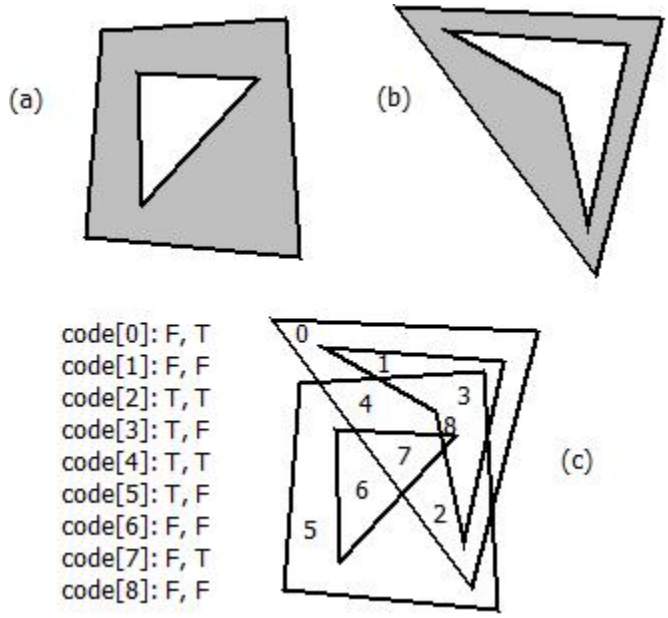

Figure 3.8: (a) $originalRegions[0]$ (b) $originalRegions[1]$ (c) All regions detected in the map, along with their codes. For example, $code[5]$ shows that region no.5 is inside $originalRegions[0]$ but outside $originalRegions[1]$.

$codedRegions[i].region$ contains the same object as $mapRegions[i]$. The value of $codedRegions[i].code[j]$ can be either $true$ or $false$. In case it is $true$, it means the entire $codedRegions[i].region$ is inside $originalRegions[j]$; and in case it is $false$, it means the entire $codedRegions[i].region$ is outside the $originalRegions[j]$. Figure 3.8 shows a simple map which is created by adding two original regions.

After all computations, the Map object uses the computed $codeRegions$ to support a wide range of queries. For example the following query returns the intersection of $originalRegions[0]$ and $originalRegions[1]$:

RETURN all $codedRegions$ WHERE $code[0] == true$ AND $code[1] == true$

Numerous simple and complex queries can be requested from a Map object which makes it a flexible geometric tool to support our localization framework. Other than the query facility, this class also has an important method which is used by our localization framework: *vector<Region> integrateOriginalRegions()*. This method "integrates" (i.e., constructs the union) of the set of all regions inside the map into the minimum possible number of regions and returns the resulting set. Figure 3.9 shows an illustration of this method's operation. This method allows us to perform expansion of non-convex polygons too, which may result in a region with holes. We first use the method $partitionIntoConvexPolygons$ to partition the non-convex polygon into the minimum possible number of convex polygons. We then expand each convex polygon separately, and finally the expanded polygons are integrated to form a region, possibly with holes. Figure 3.10 shows the steps for such an example.

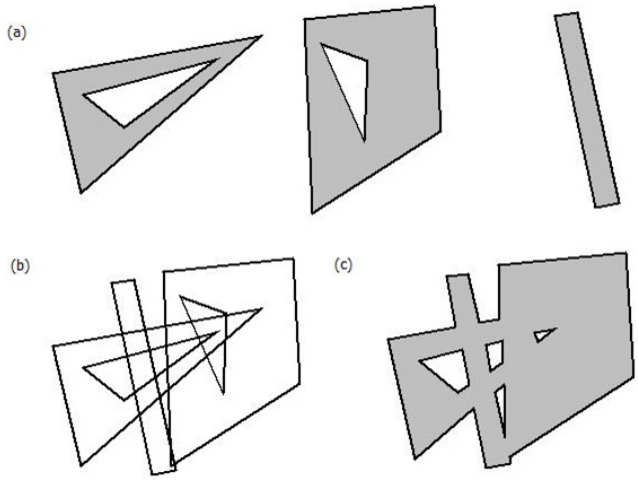

Figure 3.9: (a) Three arbitrary regions. (b) A Map object is created and regions are added to it. (c) A method in the Map class integrates all the regions inside the Map object into the minimum number of regions.

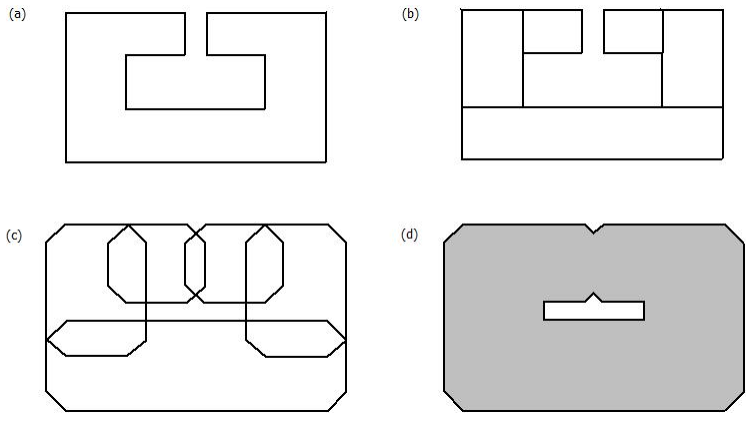

Figure 3.10: (a) A non-convex polygon to be expanded. (b) Partitioning into some convex polygons. (c) Expanding each convex polygon separately. (d) Integrating the results into a region.

Chapter 4

Indoor localization framework

4.1 Problem definition

Indoor localization, in the most general sense refers to a system employable in a set of **building environments** where a set of **sensors** record some **environmental data**; then a set of **algorithms** process that set of data to localize a **number** of **targets**. An indoor localization system has certain expectations with respect to the **timeliness** of its estimates.

There are seven bolded items in the above definition, each of which may vary in different indoor localization methods. By specifying this set of seven variants, a specific indoor localization method is well defined, thus, for our framework these seven variants have the following specifications:

Type of building environment: Any flat indoor area with ceiling height of less than 6 meters can be covered by our localization system, provided that sensors can be somehow attached, at least, at the ceiling. This 6-meter constraint is due to the limited range of PIR motion sensors that we use. Multi-storey buildings do not make any difference to our system's functionality except for stairways which are not flat areas and cannot be covered by our system.

Sensors: We have used three types of sensors: (1) passive infrared motion (PIR) sensors, (2) occupancy sensors for beds and chairs with fixed locations, and (3) switch-type sensors. By (3) we mean any sensor which has on/off or open/close states that can be changed by the person, such as magnetic reed switches and electric current sensors (determining whether current flows or not to a device). After this, almost everywhere we categorize (1) and (2) under the umbrella of binary sensors. Regarding the non-ideal behavior of the sensors in the real world, we consider two groups: the first group simplifies the problem assuming that all the sensors are ideal, while the second group tries to account for all kinds of problems caused by the sensors' imperfections. Contrary to the methods in the first group, the methods in the second group can be directly employed in a real-world application. The final product of this thesis is definitely in the second group and can be installed and run in any building.

Environmental data: Passive infrared motion sensors are triggered when any object (ideally, the target person) moves within their range. Occupancy sensors are installed in beds and chairs with

fixed locations; they are triggered when someone lays on the place. Magnetic reed switches are often installed on doors (including entrance doors, bedroom doors, fridge doors, etc.); we assume that they output 1 if the door is open and 0 otherwise. Electrical current sensors are installed on electrical switches such as light switches and in mains plugs; we assume that they output 1 if the circuit is close (i.e., current flows) and 0 otherwise.

Localization algorithms: This chapter is about describing our indoor localization algorithms in detail. Briefly speaking, our algorithms are mainly geometric and are extensively using our own geometry library which was described in the previous chapter.

Number of targets: In the current version we support the localization of one person in the building. We further assume that there is no other moving person in the environment.

Timeliness: There are two extreme points characterizing the timeliness of the localization: (a) real-time, which is representing a completely online localizer, and (b) off-line which represents a completely offline localizer, i.e., one that none of the localization estimates are produced during the observation period. A system can be either completely online (real-time) or completely offline or something in between, i.e. localizing within a delay lag, e.g. 8 seconds. Our system is almost completely real-time, except for one special case required to overcome false negative signals. We will see in the following that one solution for handling false-negative signals requires that we add one second of delay which sacrifices the real-time nature of the system. In fact, there is a compromise between the timeliness of localization estimates and accuracy. Obviously, a real-time system can be used as an offline system, but the reverse does not hold, although, in general offline systems can be more accurate due to their ability to perform extensive calculation, that are prohibitive in a real-time setting.

In section 4.2, we explain how we have modeled different entities in our localization framework. In order to simulate a building with its sensors we have developed a simulator software. In section 4.3, use cases of this simulator are briefly described. The evolution of our framework is step by step introduced through three sections: In section 4.4, our complete framework which works with only *ideal* binary sensors is explained. In section 4.5, the switch-type sensors are integrated with our system. In section 4.6, a real-world version of our framework is described. The main challenges in the real-world application are enumerated and our solutions for each of them are described.

4.2 Models

In this section, we describe how we model each of the three main entities in our localization framework: (1) the building, (2) the person (avatar in the simulations), and (3) the sensors. We also model the noise in the sensor readings. It should be noted that, geometrically the environment is modeled as a three-dimensional Cartesian system. XY plane (with $Z = 0$) represents the building's floor plane, and the positive Z-axis extends to the building height.

4.2.1 Physical model of entities

The building is a set of interconnected rooms and a set of doorways:

- Each room is modeled as a sequence of walls, a set of *light-blocking* obstacles, and a set of *non-light-blocking* obstacles in the room. A light-blocking obstacle is the one which is too tall and thus blocks the sensors' rays, e.g. a column in the middle of the room. A non-light-blocking obstacle is the one which is short enough not to block the sensors' rays, e.g. a chair or a table in the room. The sequence of walls are modeled by a polygon, where each edge of the polygon represents a wall of the room. Each obstacle is modeled by a polygon as well. The person cannot walk through an obstacle regardless of its type. Obviously, each wall has the same impact on limiting the sensor "view" as any light-blocking obstacle.

- Each doorway is modeled as a polygon, usually a rectangle. A doorway has an anti-blocking impact. It means, if a part of a wall is covered by a doorway's polygon, that part of the wall is actually "destroyed" and no longer blocks the sensors' view.

We model the person's body as a cylinder with height $bodyH$ and base radius $bodyR$. These two are settable parameters. However, $bodyR$ should be almost half the length a person's normal stride. The walking speed of the person is also represented by $walkingSpeed$ which is a settable parameter.

We model three types of sensors: (1) PIR sensors, (2) occupancy sensors, and (3) switch-type sensors. The signal of a type (1) or type (2) sensor is stored in a single bit which indicates whether the event (assumed to be associated with the activity of a person) is detected or not. The status of a type (3) sensor is also stored again as a bit to indicate the on/off or open/close status. The physical model of each sensor follows:

- *PIR motion sensors*: These sensors have a pyramid-shaped sensing range. The base of the pyramid is a rectangle. We use the spot-type motion sensor [1] for which the pyramid's height, base length, and base width are respectively $5m$, $5m$, and $3.5m$. This sensor can be attached to a wall or installed under the ceiling. We may also tune its orientation to have the desired sensing range.

- *Occupancy sensors*: Occupancy sensors are usually installed inside a beds and chairs with fixed locations. They are triggered when the person sits on the area. We model each occupancy sensor as a polygon which shows its sensing area. In the example of a bed, the entire bed would be the sensing area (assuming there is only one switch triggered regardless of where the person rests on the bed).

[1] The assumed specifications for PIR motion sensors are consistent with a typical motion sensor, e.g., a pyroelectric NaPiOn sensor (such as the Panasonic AMN43121 and similar models http://pewa.panasonic.com/assets/pcsd/catalog/napion-catalog.pdf).

- *Switch-type sensors*: Examples of switch-type sensors in real-world are magnetic reed switches and electric current sensors. We model all such sensors as a general type of switch-type sensors. A switch-type sensor is defined by a point showing the sensor's position called $switchPos$ and a polygon which shows the area in which the person has to be present in order to can change the switch. This polygon is called the sensor's $bodyRange$.

4.2.2 Noise model

In order for our simulations to resemble the real-world with reasonable fidelity, we need to simulate the noise, imperfections, and malfunctions of the sensors too. Yet, we need to keep it as simple as possible, so we have considered two distinct models: (1) *Ideal model* where all sensors are ideal with no noise or malfunction, and (2) *Realistic model*. We have closely observed the performance of different types of sensors in the real-world. According to those observations, the malfunction and noise rates of switch-type sensors and occupancy sensors are almost zero; so in our framework, the realistic model of these two sensor types is equal to their ideal model. But for passive infrared motion sensors, four factors contributing to signal noise are modeled: (a) detection pattern, (b) signal pattern, (c) false-negative occurrence, and (d) false-positive occurrence.

- *Detection pattern*: In the realistic model, a motion sensor does not detect a stationary person even if it is within its sensing range.

- *Signal pattern*: When a motion sensor is not detecting the person, the signal is a constant 0. But, in case of detection, the signal is not a constant 1. It usually alternates between 0 and 1. The time period of this oscillation depends on the update period of the motion sensor and, of course, on the movement of the subject within the area of coverage. In our realistic model, the update period of the motion sensors is 0.5 seconds, so every 0.5 seconds a transition in the signal could occur. Any sequence of transitions (regardless of how long this sequence is) where each $1 \rightarrow 0$ ($0 \rightarrow 1$) and its subsequent $0 \rightarrow 1$ ($1 \rightarrow 0$) transition are no more than 0.5 seconds apart are regarded as an oscillation

- *False-negative occurrence*: In our realistic model, a true positive signal can turn into a false-negative signal with probability P_{fn} which is computed by the following formula:

$$P_{fn} = \begin{cases} P_{base} & \text{if } fn_{-1} = 0 \\ P_{rep} & \text{otherwise} \end{cases} \tag{4.1}$$

  where $P_{rep} \geq P_{base}$, and fn_{-1} is 1 if the previous signal has been false-negative, and is 0 otherwise. In simpler words, when the true signal is positive, with probability P_{base} it may turn into a false-negative signal. But, if the previous signal has been false-negative, this probability increases to P_{rep}. This simulates the burstiness of false-negative signals that we have observed in the real-world. In the default realistic model, $P_{base} = 0.15$ and $P_{rep} = 0.5$. We have modeled this function and set these numbers intuitively according to our observation of the real sensors' behavior.

- *False-positive occurrence*: In real motion sensors, false-positive signals happen so rarely that we have neglected it in our realistic model.

4.3 Simulator

The simulator simulates a real environment to provide a sequence of data called an *event trace*. An event trace is a sequence of *event tuples* which contain time-stamped data from sensors in the same format as the real sensors would generate. Formally, each event tuple is in the form of $(code, t)$ which means the status of sensors at time t has been $code$, where $code$ is an array of bits such that $code_i$ shows the status of the i-th sensor. The process of generating an event trace requires a preparatory step, the generation of an actual location-time trace of the person. This trace is called a *raw trace* which is a sequence of *raw tuples*. A raw tuple is in the form of (p, t), which means the person's center of body mass has been in point p at time t. In order for the simulator to generate a raw trace, it requires to input the person model and the building model. After the simulator's start, the clock starts running and the user can use an avatar to traverse the space by mouse-clicking on the desired destination point. Concurrently, a new raw tuple is generated every $rawCycle$ seconds. By default, $rawCycle = 0.5secs$, but it can be set to any number prior to the start. It should be noted that a raw trace is independent of sensor readings, that is why no sensor model is loaded in this step. Once the user terminates the simulator, the generated raw trace is stored in a file which can be later used to generate various *event traces*.

To generate an event trace out of a raw trace, the simulator requires two additional inputs: the sensor configuration, and the noise model. In this mode, the simulator iterates on the sequence of raw tuples in the raw trace. For each raw tuple (p, t), it takes into account all the models to check if the person's body has touched any sensor's sensing range. And according to the noise model it decides about the signal of each of the sensors at time t. In this way an event tuple $(code, t)$ is generated out of a raw tuple (p, t). By iterating over the entire raw trace, an event trace is generated. The generated event trace is the final product of the simulator which is later directly input to the localizer.

4.4 Localization using ideal binary sensors

In this section, we explain the first phase of our localization algorithm which is only using ideal binary sensors. In this phase, we simply set $bodyR = 0$, meaning that the person's body is modeled as a vertical line with a settable height along the Z axis (e.g. $bodyH = 180cm$). So, in this phase, the projection of the person's body on XY plane is just a single point; so the person's body has no thickness and $bodyR = 0$. In next two sections, we will proceed to integrate switch-type sensors as well as to enable our system to work in real-world conditions with non-ideal sensors and a more realistic model of the person's body too. The following pseudo-code step by step shows how the

localizer works in this phase:

```
1: function LOCALIZER(BuildingModel, SensorConfiguration, PersonModel, NoiseModel)
2:     PRECOMPUTE-GEOMETRIC-MODELS
3:     LOCALIZATION-LOOP
4: end function
5: function PRECOMPUTE-GEOMETRIC-MODELS
6:     for i = 0 → ||rooms|| − 1 do
7:         rooms[i].SR ← COMPUTESENSINGRANGES(rooms[i])
8:         rooms[i].map ← COMPUTESENSINGRANGEMAP(rooms[i], rooms[i].SR)
9:         globalCodedRegions[i] ← UNIFYCODES(rooms[i].map.codedRegions)
10:        BUILDING.CODEDREGIONS.APPEND(globalCodedRegions[i])
11:    end for
12: end function
13: function LOCALIZATION-LOOP
14:    while system is running do
15:        eventTuple ← GETNEXTEVENTTUPLE
16:        probableRegions ← FINDPROBABLEREGIONS(eventTuple)
17:        prunedPRs ← PRUNEPROBABLEREGIONS(probableRegions)
18:        guessPoint ← COMPUTESINGLEPOINT(prunedPRs)
19:        STORERESULTS(prunedPRs, guessPoint)
20:    end while
21: end function
```

The localizer has two main stages: (a) pre-computation stage and (b) a real-time loop. In the pre-computation stage, three steps are performed for each room: (1) In line 7 for a given room, the sensing range of each binary sensor in the room is computed on the XY plane. (2) In line 8 for a given room, using the Map class introduced in chapter 3, a map of all computed sensing ranges is built. (3) In lines 9 and 10 for a given room, the local codes[2] (valid only in the room) of regions detected by the computed map are converted to globally (building-wide) valid codes and the resulting coded regions are added to the list of all sensing coded regions in the building. Each of those three steps are explained in detail in three following subsections. After the pre-computation is done, the real-time loop of localization starts. In each iteration, a new event tuple is received. Then three steps are done to compute an updated location of the person: (1) In line 16, according to the status of sensors in the event tuple, regions which are probable to include the person at the moment are found. (2) In line 17, according to the person's last-updated position and the time difference between the current and the previous event traces, regions found in the previous step are pruned. (3)

[2]We introduce here the concept of a "code" which is elaborated upon later. Suffice is to say for the time being that the code is a representation of the sensors' value vector.

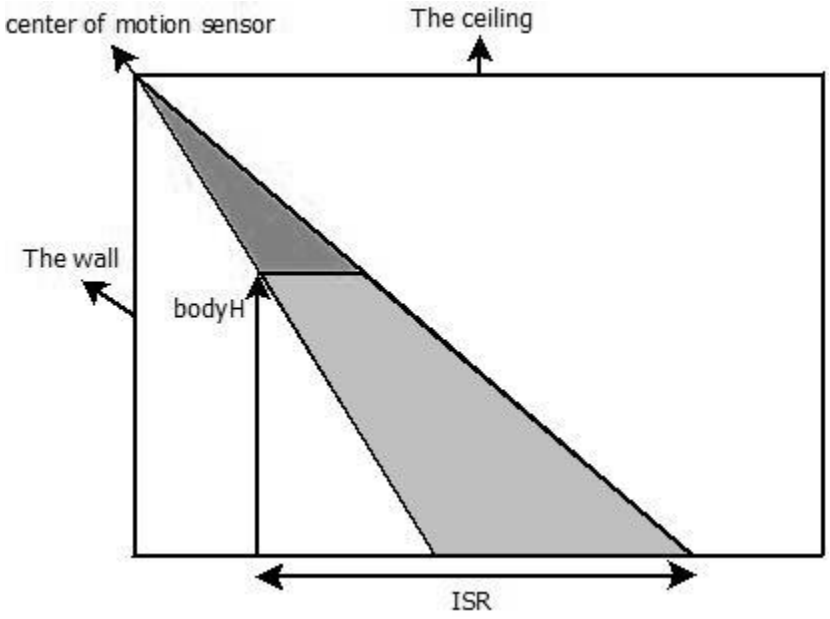

Figure 4.1: A side view of a motion sensor's pyramid. The lighter part of the pyramid contains those points of it whose height is less than $bodyH$. The projection of this lighter part on the floor forms the motion sensor's ISR.

In line 18, finally the single-point guess of the person's position is computed. The three steps in the pre-computation stage and the three steps in the real-time loop stage are explained in detail in the six following subsections.

In this thesis, the term $rooms[i]$ refers to the i-th room in the building. Also, in this section $rooms[i].sensors[j]$ refers to the j-th binary sensor inside the i-th room, and $building.sensors[i]$ refers to the i-th binary sensor inside the entire building.

4.4.1 Computing binary sensor ranges (pre-computation)

A binary sensor in our work can be either a motion sensor or an occupancy sensor. Occupancy sensors are often installed in beds and chairs with fixed locations. Hence, the range of an occupancy sensor is not a 3-dimensional volume, but is actually a 2-dimensional surface. We can therefore define their range as 2-dimensional regions in XY plane. But, a motion sensor has a 3-dimensional pyramid-shaped range in XYZ space. To simplify our next computations and algorithms, it is desirable to reduce the 3-dimensional model of their sensing ranges in XYZ space to a 2-dimensional model in XY plane. For this purpose, according to the exact position and the orientation of the pyramid, we need to compute the set of all points in XY plane such that if the person stands straight on them, at least one point of the person's body enters the pyramid and triggers the sensor. This set of points form a region in XY plane which we call the initial sensing range (ISR) of the sensor. We used the term "initial", because we have not yet taken into account the walls and obstacles which block the rays of motion sensors. A motion sensor's ISR is computed by projecting those points of

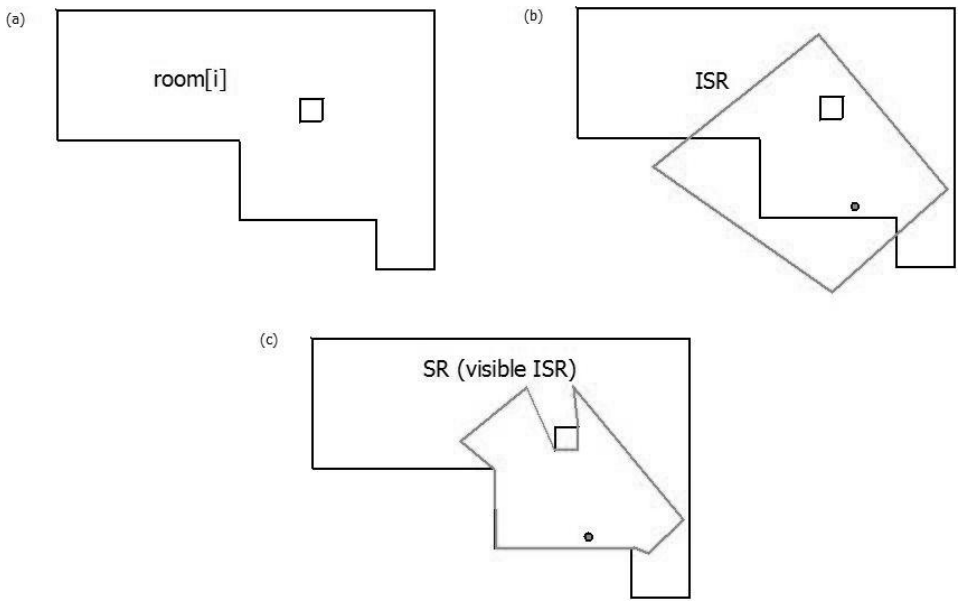

Figure 4.2: (a) Map of the room from the top view; the small square in the middle of the room is a column. (b) Initial sensing range (ISR) of a motion sensor whose center is shown by a small circle. (c) The final visible sensing range of the motion sensor (SR), after taking into account all obstacles.

the sensor's pyramid (in XYZ space) whose $Z < bodyH$ on the XY plane. Figure 4.1 shows the side view of a motion sensor's pyramid and how ISR is computed. Finally, the sensing range (SR) of a motion sensor is computed out of ISR. SR is actually a subset of ISR which is visible from the projection of the sensor's center on XY plane. This computation is performed by our geometry library. Figure 4.2 shows steps of computing a motion sensor's SR.

Now that the range of both types of binary sensors are modeled in XY plane, we can collect the sensing region of all binary sensors of the i-th room in a single array of regions : $rooms[i].SR$. Obviously, the length of this array is equal to the number of binary sensors in the i-th room and $rooms[i].SR[j]$ is a region showing the sensing range of the j-th binary sensor in the i-th room.

4.4.2 Building maps and unifying codes (pre-computation)

After $rooms[i].SR$ is computed for all rooms, we compute $rooms[i].map$ for each room. $rooms[i].map$ is an instance of the Map class. We simply add all regions in $rooms[i].SR$ to $rooms[i].map$ and as described in section 3.3 the map automatically finds an array of CodedRegion objects called $rooms[i].map.codedRegions$. $rooms[i].map.codedRegions[j].code[k]$ is a *boolean* value which says if the j-th region in the map is inside $rooms[i].SR[k]$ (i.e. inside the sensing range of the k-th binary sensor in the i-th room). Figure 4.3 shows a room with four binary sensors. For simplicity the room as well as the sensing ranges are all rectangular in this example. $\|rooms[0].SR\| = 4$ and $\|rooms[0].map.codedRegions\| = 10$. The list of codes (representing the vector of values of the binary sensors) is as follows:

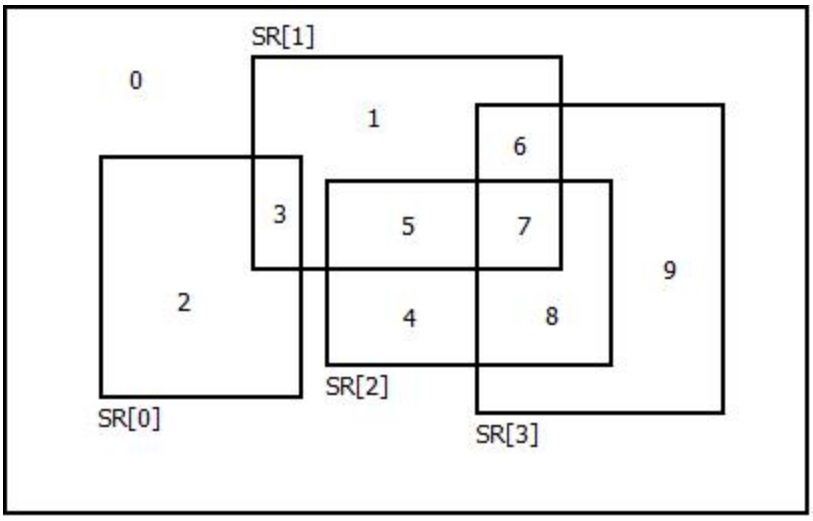

Figure 4.3: A map built for a rectangular room with four binary sensors. $\|rooms[0].SR\| = 4$ and $\|rooms[0].map.codedRegions\| = 10$.

$$rooms[0].map.codedRegions[0].code = \{0, 0, 0, 0\}$$
$$rooms[0].map.codedRegions[1].code = \{0, 1, 0, 0\}$$
$$rooms[0].map.codedRegions[2].code = \{1, 0, 0, 0\}$$
$$rooms[0].map.codedRegions[3].code = \{1, 1, 0, 0\}$$
$$rooms[0].map.codedRegions[4].code = \{0, 0, 1, 0\}$$
$$rooms[0].map.codedRegions[5].code = \{0, 1, 1, 0\}$$
$$rooms[0].map.codedRegions[6].code = \{0, 1, 0, 1\}$$
$$rooms[0].map.codedRegions[7].code = \{0, 1, 1, 1\}$$
$$rooms[0].map.codedRegions[8].code = \{0, 0, 1, 1\}$$
$$rooms[0].map.codedRegions[9].code = \{0, 0, 0, 1\}$$

Codes inside $rooms[i].map.codedRegions$ are local codes inside the i-th room with the length of $\|rooms[i].sensors\|$. In the next step, we make $building.codedRegions$. Regions inside $building.codedRegions$ are the union of all regions inside $rooms[i].map.codedRegions$ for all i's. But, the code inside each CodedRegion object is converted from the local format (valid inside the room) to the global format (valid inside the entire building). After this globalization of codes, all codes in $building.codedRegions$ have the same length of $\|building.sensors\|$.

4.4.3 Finding probable regions (real-time computation)

So far, we have had three steps of computation all of which are performed prior to the real-time running of the system (i.e. the localization loop): (1) computing binary sensor ranges, (2) building maps, and (3) unifying codes. This subsection, describes the next step which is the first of three steps in the localization loop. The input to this step is a new event tuple which is just received. As we explained in section 4.3, an event tuple is in the form of $(code, t)$ which means the status of binary

sensors at time t has been $code$, where $code$ is an array of bits such that $code_i$ shows the status of the i-th binary sensor. For example suppose the room in figure 4.3 be the only room in the building. So if sensors in the figure are all ideal, the presence of the person in region 5 at a given time would make an event tuple whose $code = \{0, 1, 1, 0\}$. The following terms will also be used throughout the rest of this thesis:

- $tuple_{-1}$ refers to the previous event tuple. $tuple_0$ refers to the current event tuple. t_{-1} refers to the time-stamp of $tuple_{-1}$ and t_0 refers to the time-stamp of $tuple_0$.
- $realPos[t]$ is a two-dimensional point which shows XY coordinates of the *actual* position of the target person's center of mass at time t. $realPos$ with no specified time means $realPos[t_0]$.

In this step, the localizer searches into the array $building.codedRegions$. It collects every coded region whose *code* exactly matches $tuple_0.code$. The regions of all such coded regions are collected in an array of regions called "initial probable regions" at time t_0 ($initialPRs[t_0]$).

4.4.4 Pruning probable regions using short history (real-time computation)

$initialPRs[t_0]$ is a *maximal* set of regions which may possibly contain the target person at time t_0. By *maximal* we mean that the target person is *certainly* in one of those regions, but we may still prune parts of those regions and narrow down the probable regions; the result is an array of regions called "pruned probable regions" at time t_0 ($prunedPRs[t_0]$) which is computed as follows:

We compute $prunedPRs[t_0]$ out of three inputs: (1) $initialPRs[t_0]$, (2) $prunedPRs[t_{-1}]$, and (3) the time difference between t_0 and t_{-1}, i.e. $(t_0 - t_{-1})$. It should be noted that at the start, since there is no t_{-1}, there is no pruning step either. Depending on the application and the specific person that we are localizing, we can consider a value for the person's normal walking speed. By *normal* speed, we specifically mean the average speed over 25% fastest moments. We call this value as $walkingSpeed$. If the person does not walk surprisingly faster than $walkingSpeed$, then logically, at time t_0, the person could not have deviated from borders of $prunedPRs[t_{-1}]$ by more than $maxDeviation$ meters which is simply calculated in the following equation:

$$maxDeviation = walkingSpeed * (t_0 - t_{-1}) \tag{4.2}$$

Thus, the three stages below are followed:

1. A parameter called $expDist$ (standing for "expansion distance") is calculated as follows:

$$expDist = maxDeviation * expCoef \tag{4.3}$$

   where $expCoef$ is a real number usually between 0.5 and 2 which should be tuned. By default, $expCoef = 1$.

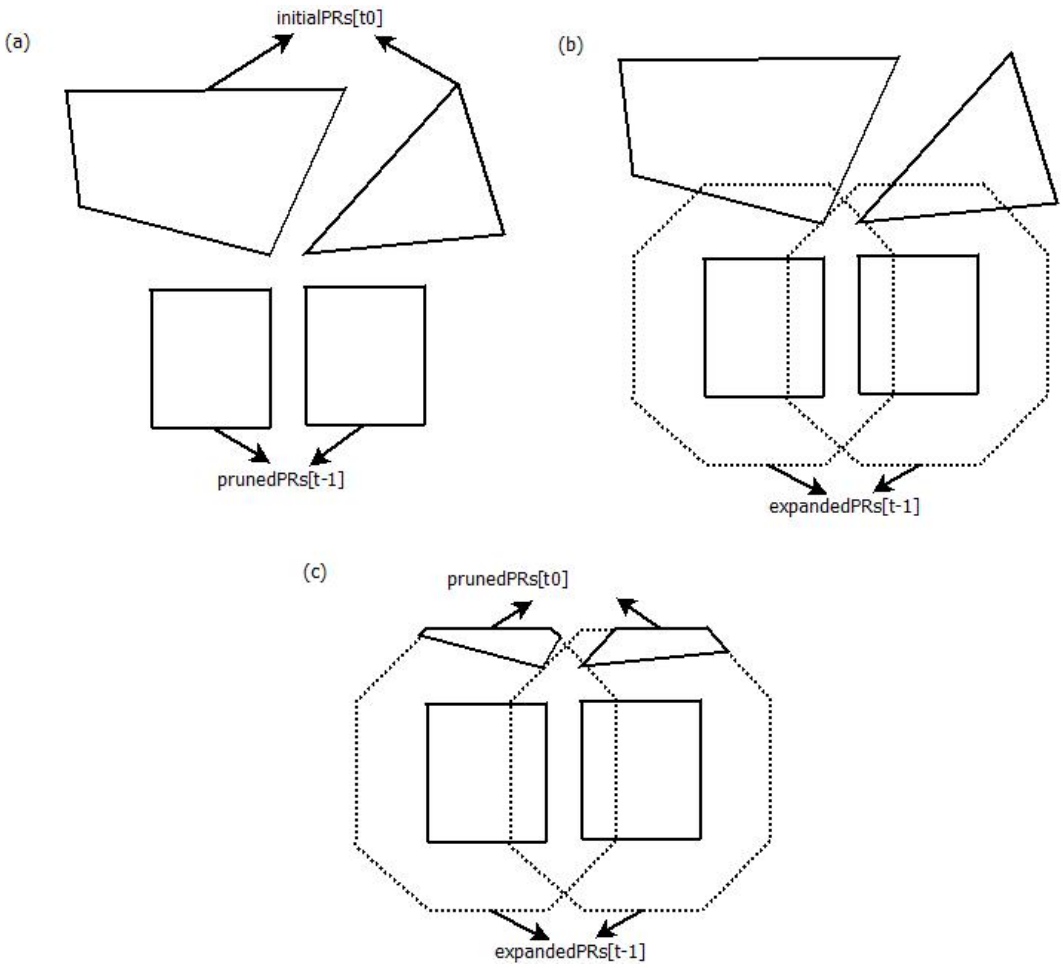

Figure 4.4: (a) In step 1, we have two arrays of regions: $initialPRs[t_0]$ and $prunedPRs[t_{-1}]$. (b) $expDist$ is computed and all regions inside $prunedPRs[t_{-1}]$ are expanded by $expDist$ units of distance to obtain $expandedPRs[t_{-1}]$. (c) Finally, the intersection of $prunedPRs[t_0]$ and $expandedPRs[t_{-1}]$ results in $prunedPRs[t_0]$.

2. Second, we expand all regions inside the array $prunedPRs[t_{-1}]$ by $expDist$ units of distance to generate another array named $expandedPRs[t_{-1}]$.

3. Finally, $prunedPRs[t_0]$ is calculated by intersecting two sets of regions:

$$prunedPRs[t_0] = initialPRs[t_0] \cap expandedPRs[t_{-1}] \tag{4.4}$$

   The intersection of two or more sets of regions is calculated by our geometry library in linear time complexity in the total number of regions' vertices.

4. If $prunedPRs[t_0] = \emptyset$, then increase $expCoef$ by 0.1 and go to the step 1.

If everything is ideal, i.e. (a) value of $walkingSpeed$ is not underestimated, and (b) all binary sensors are ideal and never malfunction, $prunedPRs[t_0]$ never becomes empty in the step 4. In

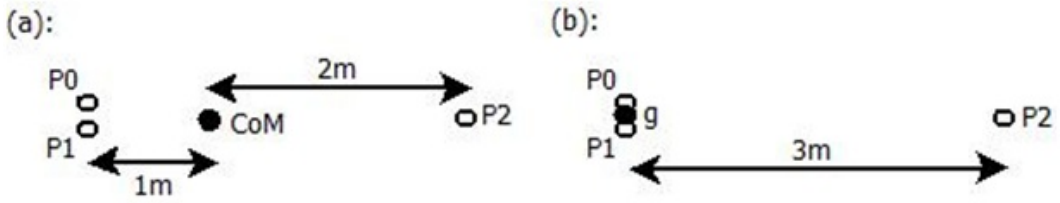

Figure 4.5: $prob(P0) = prob(P1) = prob(P2) = 1/3$. (a) The guess point is CoM which is the center of mass of the three points, so $expectedError(CoM) = 2 * (1/3) + 1 * (1/3) + 1 * (1/3) = 4/3$.(b) The guess point is g, so $expectedError(g) = 3 * (1/3) + 0 * (1/3) + 0 * (1/3) = 1$. So $expectedErr(g \neq CoM) < expectedErr(CoM)$.

fact, if $prunedPRs = \emptyset$ in the step 4, there are two possibilities: (a) The person may have moved with an abnormally high speed that exceeds $walkingSpeed$. (b) Some sensor events may have been missed or noticed with delay. In this case, a *jump* in the pace of localization might happen. Figure 4.4 shows the step by step computation of $prunedPRs[t_0]$ in an example.

4.4.5 Computing a single-point guess (real-time computation)

Up to this point, we have computed an array of regions called $prunedPRs[t_0]$ which contains all regions which can potentially contain the target person at time t_0. In the final stage of localization, we should output a single-point which shows our best guess of the target person's position. But, what we mean by the *best* point, is a point that minimizes the expected value of the error, i.e. the distance between $guessPoint[t_0]$ and $realPos[t_0]$. The expected error for a given $guessPoint$ at time t is calculated as follows:

$$expectedError(guessPoint, t) = \iint_{prunedPRs[t]} dist(guessPoint,\, dA).prob(t, dA)\, dA \qquad (4.5)$$

where $0 \leq prob(t, A) \leq 1$ represents the probability that region A contains the target person at time t. One simple heuristic to compute $guessPoint[t_0]$ is:

$$guessPoint[t_0] = prunedPRs[t_0].centerOfMass \qquad (4.6)$$

Center of mass of an area is the average of the coordinates of all points in that area. Using the center of mass of the probable regions as $guessPoint[t_0]$ has two problems:

- Points inside $prunedPRs[t_0]$ do not actually have the same probabilities of being $realPos[t_0]$. But, this solution does not consider this fact.
- Even if we assume the probability of containing the actual person at time t_0 is evenly distributed over all points of $prunedPRs[t_0]$, it can be mathematically proven that the center of mass of probable regions does not minimize the expected error. In case of even probability distribution, $expectedError$ function is simplified as below:

$$expectedError(guessPoint, t_0) = \iint_{prunedPRs[t]} dist(guessPoint,\, dA)\, dA \qquad (4.7)$$

A counter example shown in figure 4.5 proves the non-optimality of the center of mass even in case of even probability distribution.

Although as we proved the center of mass of the probable regions does not necessarily minimize the expected error, it is still a very good approximation of the optimal answer and also very simple to compute. Therefore, $guessPoint[t_0]$ is simply computed as below:

$$guessPoint[t_0] = prunedPRs[t_0].centerOfMass \tag{4.8}$$

4.5 Including switch-type sensors

Now that we have explained everything about ideal binary sensors, we can easily include the information that we occasionally receive from switch-type sensors. As described in section 4.2.1, a region named $bodyRange$ is assigned to every switch-type sensor. In our model, this has an approximate size of 0.5×0.5 located close to $switchPos$. When a transition happens in the output signal of a switch-type sensor, the person must be normally in the sensor's $bodyRange$. This region is made in a way that its center has the most probability of being the target person's center of mass at the moment of transition. It should be noted that a switch-type sensor's $bodyRange$ quite often does not even contain $switchPos$. For example, suppose a sensor is installed on the door of a fridge. A transition happens in its output signal when the person opens or closes the door; obviously, at this moment the person is not standing right on the sensor which is installed on the door. Normally he must have a distance of $0.4m$ with the door and tend to the same side as the door opens. This is the way we hardcode $bodyRange$ for each switch-type sensor. Having this region for every switch-type sensor, it is easy to include the information from these sensors with our previous system: Whenever we detect a transition in a switch-type sensor's signal, we assign its $bodyRange$ to $initialPRs[t_0]$, and the rest of stages (pruning, etc.) are the same as explained before. Since, switch-type sensors are highly reliable, we completely trust their signals; so, we did not need to add any intermediate stage to check if the signal is reasonable and correct.

4.6 Localization using real binary sensors

In section 4.4 we explained all stages of localization using binary sensors, on the condition that all sensors are ideal and work perfectly. But, those algorithms alone do not work in a real-world scenario. In this section, we adjust those algorithms to make a robust system which can work in real world with non-ideal binary sensors. In order to achieve this goal, we have to resolve five main problems which are caused by imperfection of the real sensors and the real environment. The imperfections generate *incorrect* sensor readings, i.e. incorrect $tuple_0.code$. An incorrect code can be either *invalid* or *misleading* which are defined as follows:

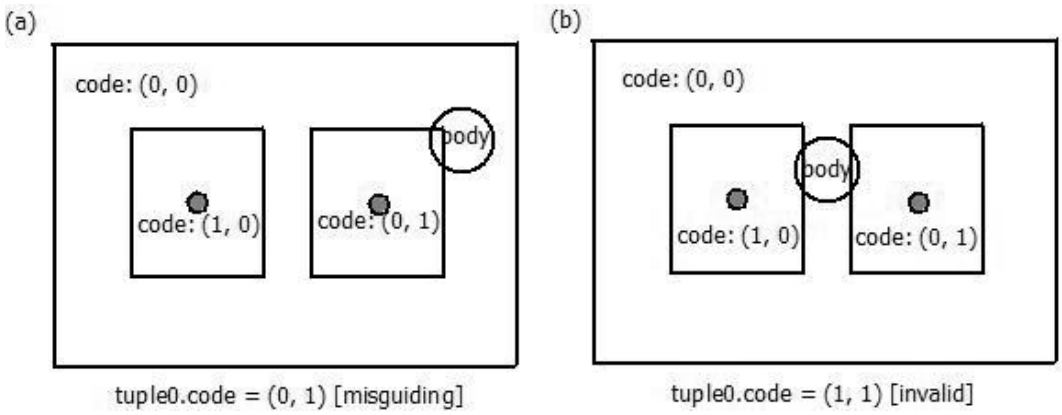

Figure 4.6: Two sensors are installed in a room. The center of each sensor is shown with a small circle and a rectangle shows the range of each sensor. In both cases the person's body (the circle) has entered a region while $realPos$ (center of the circle) is not inside that region. (a) $realPos$ belongs to a region whose code is $(0, 0)$, so $(0, 1)$ in an incorrect code, and since there is a region with the same code, this code is *misleading*. (b) $realPos$ belongs to a region whose code is $(0, 0)$, so $(1, 1)$ is an incorrect code, and since there is no region with the same code, this code is *invalid*.

Definition 1. *$tuple_0.code$ is **invalid** is a sensor reading that does not match any code inside $building.codedRegions$.*

Definition 2. *$tuple_0.code$ is **misleading** is a sensor reading that matches a coded region in $building.codedRegions$ whose region does not include the actual $realPos[t_0]$.*

With ideal sensors and a perfect environment we always have *correct* codes. But, in this section we introduce five main sources of incorrect codes (either misleading or invalid) in the real-world. We then explain how we address each of these five anomalies in separate subsections.

4.6.1 Body thickness

So far we have considered the person's body as a cylinder with normal height (e.g. 180 cm) but with radius of 0. In fact, the projection of the person's body on XY plane has been a single point in section 4.4. Now, we will consider a reasonable radius – called $bodyR$ – for the body cylinder according to a normal body size and step length. In figure 4.6 two examples are shown where the person's body thickness causes misleading and invalid codes in the previous model. We claim without proof that modeling a person's body as a thick cylinder (i.e. $bodyR > 0$) is equivalent to expanding all sensing ranges on the XY plane by $bodyR$ units of distance. So, we claim:

$$\begin{gathered}\text{“}\texttt{Taking into account } bodyR = nonZeroR\text{”} \\ \equiv \\ \text{“}\texttt{The previous body model } (bodyR = 0)\text{”} \\ +\text{“}\texttt{Expanding sensing ranges by } nonZeroR\text{”}\end{gathered} \tag{4.9}$$

So, our solution is simply this: in the first stage of pre-computation (computing sensor ranges), we expand all regions inside the array $rooms[i].SR$ by $bodyR$ units. Figure 4.7 shows how this

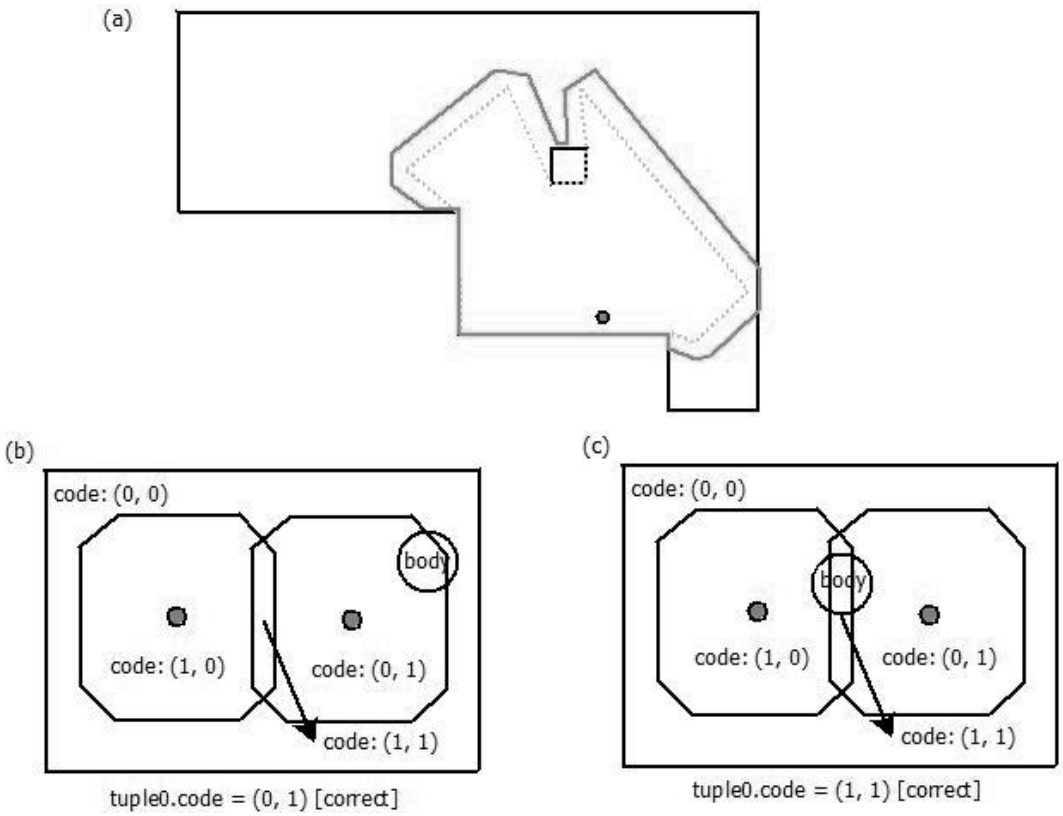

Figure 4.7: (a) Expanded sensing range of the motion sensor in figure 4.2. (b) After expanding the sensing range of figure 4.6.a $realPos$ belongs to a region whose code is $(0, 1)$ which is the same as $tuple_0.code$. So a correct code is read from sensors. (c) Following figure 4.6.b, after expansion of sensing ranges, a new region with the code $(1, 1)$ is generated, so the sensor signals are no longer incorrect at this moment.

expansion happens in the examples of figure 4.6. This clearly explains how the body thickness problem is addressed.

4.6.2 No-motion no-trigger

In the model of ideal binary sensors, the sensor's signal is 1 if and only if $realPos$ is inside the sensing range. But, in the real-world if the person is inside the sensing range but not moving, the sensor is not triggered. So, the incorrect all-zero code frequently occurs in practice. To solve this problem, whenever we observe an all-zero code, we replace it with the most recent valid non-zero code. The logic behind this decision is because the person has been moving for a while when correct codes have been generated and then he has stopped at a point after which all-zero codes are generated. So, the most recent non-zero code represents the place the person has last stopped.

4.6.3 Oscillating signals

In practice, motion sensors do not output a constant 1 signal when they sense a moving object, but an oscillating signal will appear on the output. There are two solutions:

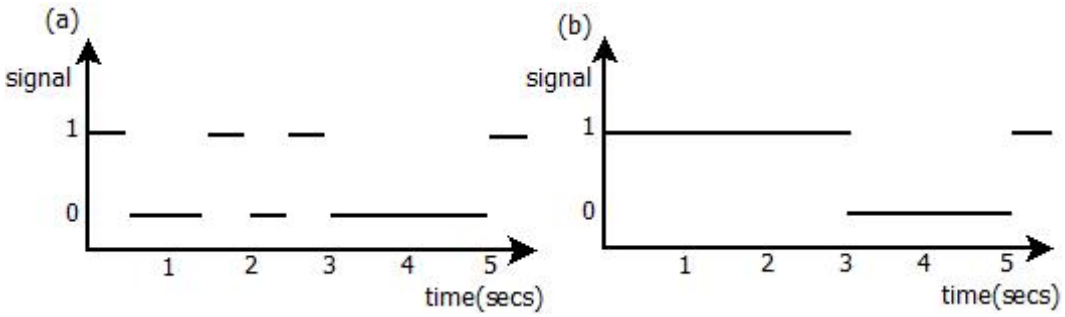

Figure 4.8: (a) original received signal over time. (b) Enhanced signal by the first approach: sacrificing responsiveness to maintain accuracy.

- The first solution sacrifices real-time localization to enhance accuracy. This is shown in figure 4.8 in an example. We actually convert each 0 value of shorter than one second to 1. For this purpose, we logically need one second of delay to determine the actual signal at a moment.

- The second solution sacrifices accuracy to maintain real-time operation. In this approach, when a transition from 1 to 0 happens in the input signal, we still consider it as a 1 signal for one more second. This way we actually extend the life of any 1 signal for one more second. To achieve this signal enhancement, we can still be real-time and no delay is required to be added. On the other hand, since not all zero-to-one transitions are oscillations and some of them are real transitions, the calculated signal occasionally becomes a false-positive.

We just use the first solution in our framework. So, to maintain accuracy, our localization has one second of delay.

4.6.4 Doors

So far we have assumed that all doors are closed all the time in the sense that the sensing range of a sensor in a room can never penetrate other rooms through doors. This assumption works perfectly in simulations where we can decide about the world's configurations. But in the real-world, doors are not guaranteed to be closed all the time, and when they are open they do not block rays to the sensor. This can potentially make cause invalid and misleading codes. Figure 4.9 shows a simple example with two rectangular rooms with a small connecting door. Figure 4.9.a shows the map of the building with all coded regions made by sensing ranges. In the pre-computation stage of localization, this map is computed since the door is assumed to be closed all the time. But if the door is open, the map in figure 4.9.b represents the actual map of the regions in the building. Now, assume a real-world scenario where the door is open and the person walks in the region A in figure 4.9.b. If both sensors work perfectly, $tuple_0.code$ will be $(0, 1)$; Although this code is actually correct and sensors have worked well, this is counted as an incorrect code since in the real map of the building shown in figure 4.9.a the code for region A is $(0, 0)$. And since there is already at least one region with code

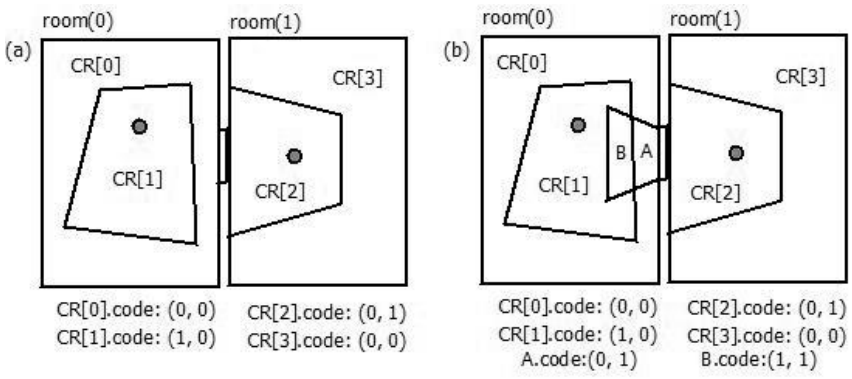

Figure 4.9: In each of the two rooms a motion sensor is installed. CR stands for $codedRegions$. (a) shows the map of regions made by sensing ranges when the door is closed; here $\|codedRegions\| = 4$. (b) shows the same map when the door is open; here $\|codedRegions\| = 6$. In the ideal model, map (a) is used; so if the door is open and $realPos$ is inside A, then $tuple_0.code = (0, 1)$ but this code belongs to $CR[2]$ which does not include $realPos$; therefore the code is **misleading**. Also, if the door is open and $realPos$ is inside B, then $tuple_0.code = (1, 1)$ but this code is not in the ideal model (a); therefore the code is **invalid**.

$(0, 1)$ in our pre-computed model, this incorrect code is *misleading*. Assume a similar scenario if the person walks in the region B while the door is open. In such a case, $tuple_0.code$ becomes $(1, 1)$ which is actually correct but according to our model it is counted as an *invalid* code.

In order to resolve this problem, for every door, we assume the door is open and then for any region which is generated by a "foreign" sensor (i.e. a sensor from another room) we consider two different codes: the normal code of that region, as well as a code by canceling all 1 bits of foreign sensors. For example, in the building of figure 4.9, the original map of coded regions are pre-computed according to figure 4.9.b, where A is a region with two different codes assigned to it : $(0, 0)$ and $(0, 1)$. And B has two different codes too: $(1, 0)$ and $(1, 1)$.

4.6.5 False-negative signals

For whatever reason a binary sensor may not detect the target in its sensing range. In the case of motion sensors, it might be because of small movements by the person which are occasionally left undetected. If no sensor detects the presence of the person at the same, $tuple_0.code$ becomes all zero and it will be easily handled the same way as we handled "no-motion no-signal" case. But, in case some sensors output a false-negative signal while others correctly detect the person, an incorrect code is produced. This incorrect code can be of both possible types: *invalid* or *misleading*. For invalid codes, all we do is treat them exactly the same way as explained in "no-motion no-signal" sub-section. It is reasonable to do the same thing about misleading codes too. But, how can we detect misleading codes? Invalid codes are easily detectable, because by definition they are not in the list of valid codes in the map; but misleading codes are those incorrect codes which are still found in $building.codedRegions$. We distinguish between a valid correct code and a valid incorrect code

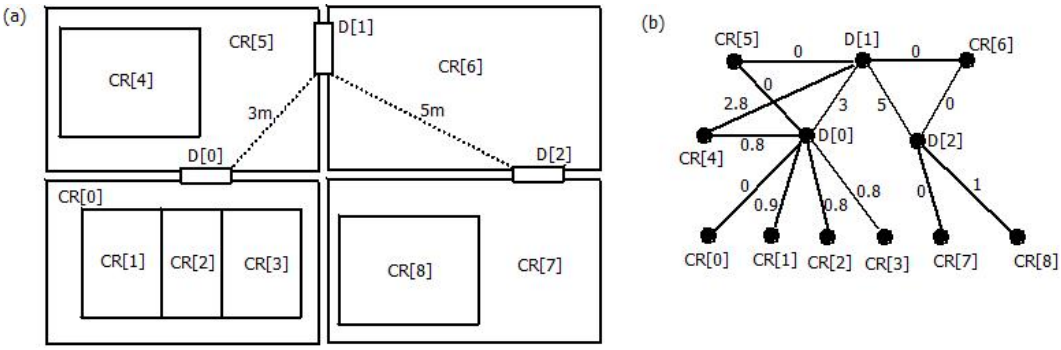

Figure 4.10: (a) The map of a building with four rooms and three connecting rooms. CR stands for $codedRegions$. There are nine coded regions in the building. The distance between two regions in the same room is calculated directly with our geometry library, e.g. $regionsDist[1][3] = 1.5m$. (b) To compute the distance between regions not in the same room, a graph is constructed. The weight of each edge is the minimum planar distance between the two nodes. According to this graph, for example the shortest *restricted* path between $CR[1]$ and $CR[8]$ is 9.9 meters. The shortest normal (non-restricted) path between these two regions would be $1.9m$ which is obviously wrong; because it illegally bypasses the $5m$ edge from $D[2]$ to $D[1]$ by going through $CR[6]$ which is connecting the two nodes with $0m$ edges. The same thing also happens from $D[1]$ to $D[0]$ by illegally going through $CR[5]$.

(i.e. misleading code) through the following procedure:

After pre-computing $building.codedRegions$ as explained before, we compute a two-dimensional array called $regionsDist$ such that $regionsDist[i][j]$ represents the minimum distance between $building.codedRegions[i].region$ and $building.codedRegions[j].region$. The minimum distance between two regions which are in the same room is simply calculated by our geometry library. But, for calculating the minimum distance between two regions in the building which are not in the same room, another solution is proposed. We construct a graph with weighted edges. Nodes of this graph are regions in the building plus doors which connect rooms. Two nodes are connected if (1) they correspond to two doors of the same room, or (2) one of them corresponds to a region in a room and the other one corresponds to a door of the same room. The weight of an edge is the planar minimum distance between the corresponding nodes: either two doors, or a region and a door. After constructing this graph, we find the shortest *restricted* path between any two nodes like n_0 and n_1 such that both nodes correspond to coded regions. We define a *restricted* path from n_0 to n_1 as a path in which a transition from a door node to a region node that is not the destination (i.e. n_1) is not allowed. The length of such a path is assigned to $regionsDist[n_0][n_1]$. Figure 4.10 shows how the graph is made for a sample building. In this figure, it is shown in an example why we have to apply this restriction for a path. All the process of computing $regionsDist$ occurs in the pre-computation stage, but in real-time this array is used to distinguish between a correct valid code and a misleading code. When a new $tuple_0.code$ appears that is valid (i.e. found in the list of codes in $building.codedRegions$), we check the minimum distance between $initialPRs[t_0]$ and $initialPRs[t_{-1}]$. If this distance is too big to be explainable according to $(t_0 - t_{-1})$ and the person's normal and maximum speed,

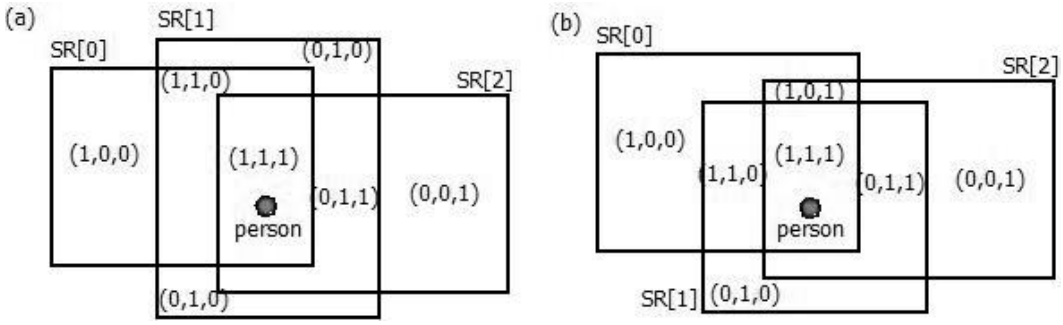

Figure 4.11: (a) Seven coded regions are created by the intersection of three sensor ranges. Assume $sensors[1]$'s signal be false-negative; so, $tuple_0.code$ is $(1, 0, 1)$. Previously, it would be considered as an invalid code. But, with the no-exclusion method, every region whose code is 1 in the first and the last bit is a probable region. So, the region which is coded as $(1, 1, 1)$ is the only probable region. (b) With this new configuration, if again $sensors[1]$'s signal be false-negative, the $tuple_0.code$ is still $(1, 0, 1)$ which is incorrect but misleading this time. Without no-exclusion method, the very small region coded as $(1, 0, 1)$ would be the only probable region. But, with the no-exclusion method, two regions with $(1, 0, 1)$ and $(1, 1, 1)$ are probable regions.

the current code is considered an incorrect code of misleading type. A detected misleading code is treated the same was as an invalid code and an all-zero code.

4.6.6 Too many false-negative signals

In the previous section, we mentioned that we treat invalid codes and detected misleading codes the same way as we treated all-zero codes in "no-motion no-signal" section. It means we replace an incorrect code with the most recent correct code that we have recorded. This is a successful approach if the frequency of incorrect codes is reasonably low. But in the case of too many false-negative signals, the current method would no longer be appropriate. In such a case, we use a method called *no-exclusion*. There is no theoretical definition for "too many false-negative signals" but in practice there is a threshold on the frequency of false-negative signals after which no-exclusion method performs better than the default setting. In experiments, we will see that this threshold depends on the sensor configuration. So far, whenever a new event tuple – i.e. $tuple_0$ – is received, probable regions are those regions whose codes exactly match $tuple_0.code$. In fact, we care about both 0 and 1 bits to decide if two codes are equal. In other words, probable regions are the intersection of those sensor ranges whose corresponding bit in the code is 1, *excluding* the other sensor ranges whose bit is 0. With our binary sensors, we have almost never experienced false-positive signals, so we are confident about 1 bits in the code. But, when false-negative signals are too frequent, it actually means that 0 bits are not as reliable. Therefore, in case of too many false-negative signals, we skip the exclusion of sensor ranges with 0 bits in the code from the probable regions. More precisely, now a region is one of the probable regions if its code is 1 in all 1 bits of $tuple_0.code$; values of the rest of its bits do not matter. By this change, the precision of probable regions is lowered, but in turn the system becomes a lot more confident and robust against false-negative signals. Figure 4.11

shows conditions in which before using the no-exclusion method, false-negative signals could lead to invalid and misleading codes with significant impact. This figure shows how the no-exclusion method cancels the effects of a significant false-negative signal where it leads to an invalid code in part (a) and a misleading code in part (b). The other positive consequence of this method is that, it makes the system 100% robust against complete malfunction or removal of binary sensors. For instance, if a binary sensor suddenly breaks while the system is running, with the logic of the no-exclusion method, this sensor is automatically removed from the computational models as if it was not existing at all. In summary, by this simple method we lose a little in localization precision, but in turn a remarkable degree of robustness is obtained.

Chapter 5

Experiments

We run all experiments in a simulated setup using the *simulator* framework. Our building model is built with little changes to the building map of the *EvAAL* indoor localization and tracking challenge that took place in Madrid in 2012. We have considered five dimensions of variability to setup the experiments:

- *Raw trace*: We create two different traces in the building. The path of both traces are the same in order for their results to be comparable. This is called the *tour* of the building which is a long enough path to cover various parts of the building. Both traces follow the tour, but they are different in terms of smooth vs. bursty movement of the person. In the smooth trace – referred to as S – the person walks at $0.7m/s$ with very few occasional stops. In the bursty trace – referred to as B – the speed is the same, but with very frequent stops on the way. Figure 5.1 shows the building model as well as its tour.

- *Sensor configuration*: We test four different sensor configurations. In two of them, only binary sensors are employed: (a) A *minimum-coverage* configuration consisting of 21 binary sensors (one occupancy sensor and 20 motion sensors). This configuration is labeled min. (b) A fairly *dense* configuration consisting of 32 binary sensors (a single occupancy sensor and 31 motion sensors). This configuration is labeled dns. By adding nine switch-type sensors to each of these two configurations, two new configurations are obtained, respectively labeled min_{swt} and dns_{swt}. Figure 5.2 and figure 5.3 show these configurations.

- *Algorithm mode I*: We test two variations: (1) Enabled pruning step as discussed in section 4.4.4, and (2) Disabled pruning step. By default we assume that the pruning step is enabled in all experiments unless marked by the symbol $Prune_{off}$.

- *Noise model*: We test five noise models:

  - *Ideal model* labeled by fn_0 ("fn" stands for false-negative. Although the noise-model is not just about false-negative signals, we traditionally use this naming.).
  - *Default realistic model* labeled fn_1 in which $P_{base} = 0.15$ and $P_{rep} = 0.5$.

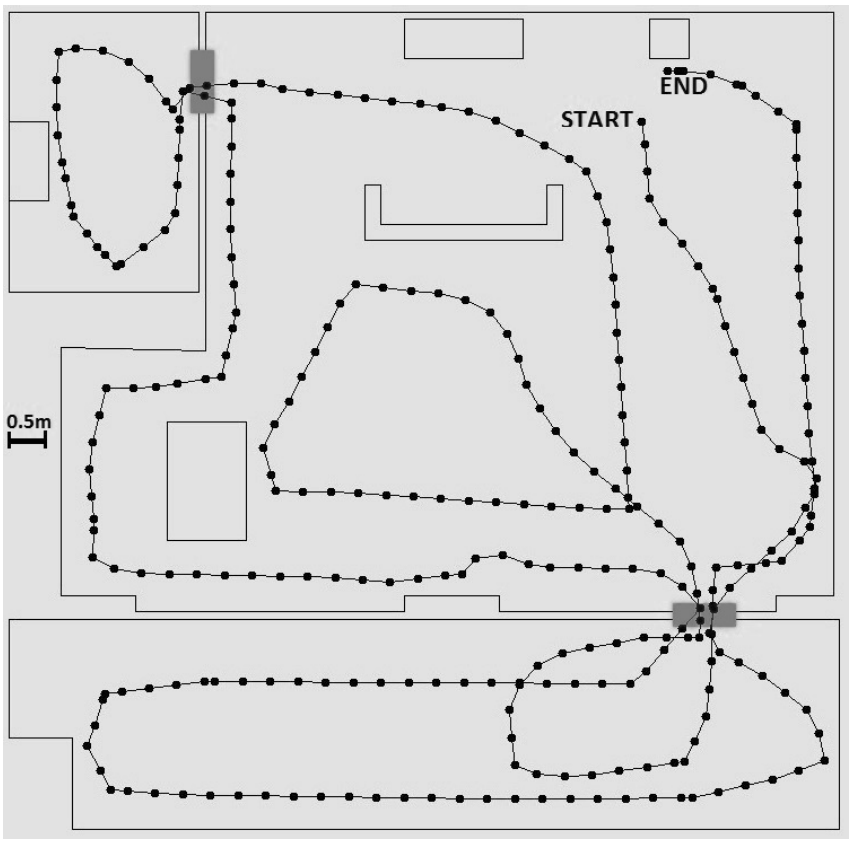

Figure 5.1: A top view of the building with three rooms: (1) The main hall which is the biggest room in the image, (2) The small washroom on the top left corner of the image, and (3) The huge balcony on the bottom of the image. These rooms are connected by two small doors. Five pieces of furniture are placed in the building: four in the main hall, and one in the washroom. The entire building is almost $120m^2$ and the ceiling height is $2.62m$. The start and end points of the tour are marked accordingly. It should be noted that the bed and any area with occupancy sensors are not considered as obstacles and the person can walk through them.

  - A non-default realistic model labeled fn_2 in which $P_{base} = 0.3$ and $P_{rep} = 0.5$.
  - A second non-default realistic model labeled fn_3 in which $P_{base} = 0.5$ and $P_{rep} = 0.5$.
  - A third non-default realistic model labeled fn_4 in which $P_{base} = 0.7$ and $P_{rep} = 0.7$.

  All three non-default realistic models are only different in the frequency of false-negative signals; they are exactly the same in other aspects of a realistic model as discussed in section 4.2.2.

- *Algorithm mode II*: We examine the effect of the no-exclusion method explained in the previous chapter. Experiments in which the no-exclusion method is enabled are marked by the symbol ex_{off} and others are marked by the symbol ex_{on}.

In the following sections we investigate the impact of each of the above dimensions on the localization error. In all experiments, the *mean* error, the *standard deviation*, and the *standard error* are

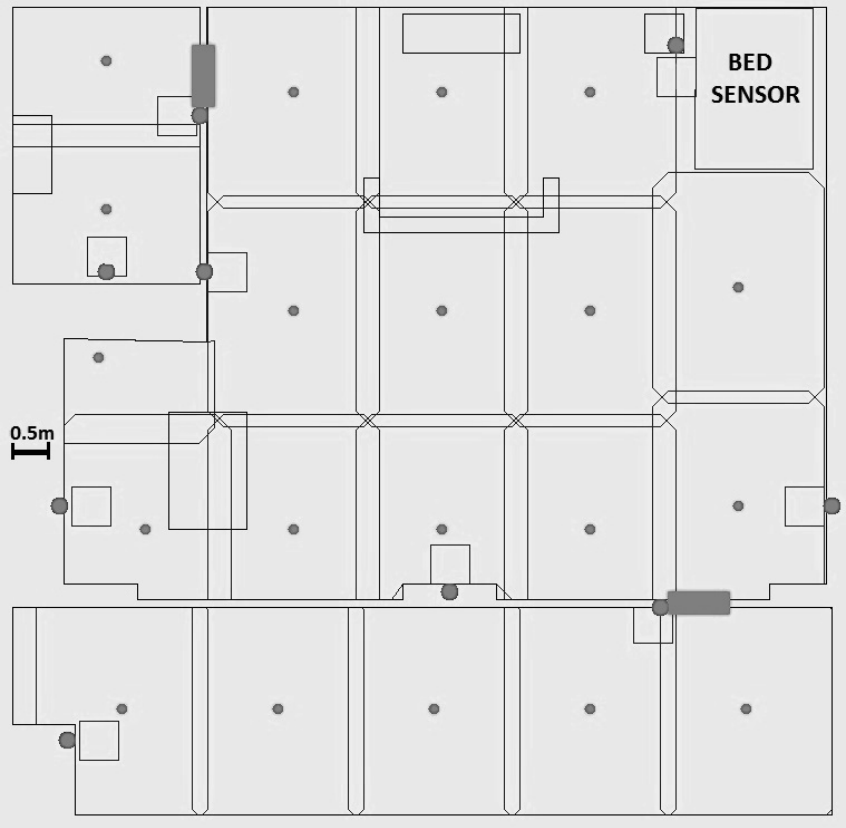

Figure 5.2: A *min-coverage* configuration of 21 binary sensors (20 motion sensors and a single occupancy sensor installed in the bed). All motion sensors are installed under the ceiling and vertically facing the floor. Small dots show the XY position of the motion sensors. The range of each sensor on the floor is expanded by $bodyR = 0.25m$ to support a much more realistic model of the person's body as discussed in section 4.6.1. Big dots show the XY position of nine switch-type sensors. Small squares next to switch-type sensors are their $bodyRange$ as introduced in section 4.2.1 and 4.5. These nine switch-type sensors are enabled in min_{swt} and disabled in min.

reported. It should be noted that, as we will see throughout the rest of this chapter, the standard deviations are of such magnitude that the overall results are not statistically comparable. Fortunately, in almost all of them, the standard error is small enough to compare the $mean$ errors.

5.1 Bursty *vs.* smooth movement

In all experiments of this section, the *no-exclusion* algorithm is disabled. In the first part, we also fix the noise model to fn_0 to investigate the impact of the smooth (S) vs. bursty (B) movements on two sensor configurations : min and dns. The results are reported in the table 5.1. In both pairs of experiments, the localization error for the smooth movement is lower than for the bursty one. This result could be expected, because the pruning algorithm considers a constant speed for the target person which is satisfied by a smooth trace while the bursty trace does not always satisfy this assumption. Another observation is that the amount of improvement for the smooth movement is

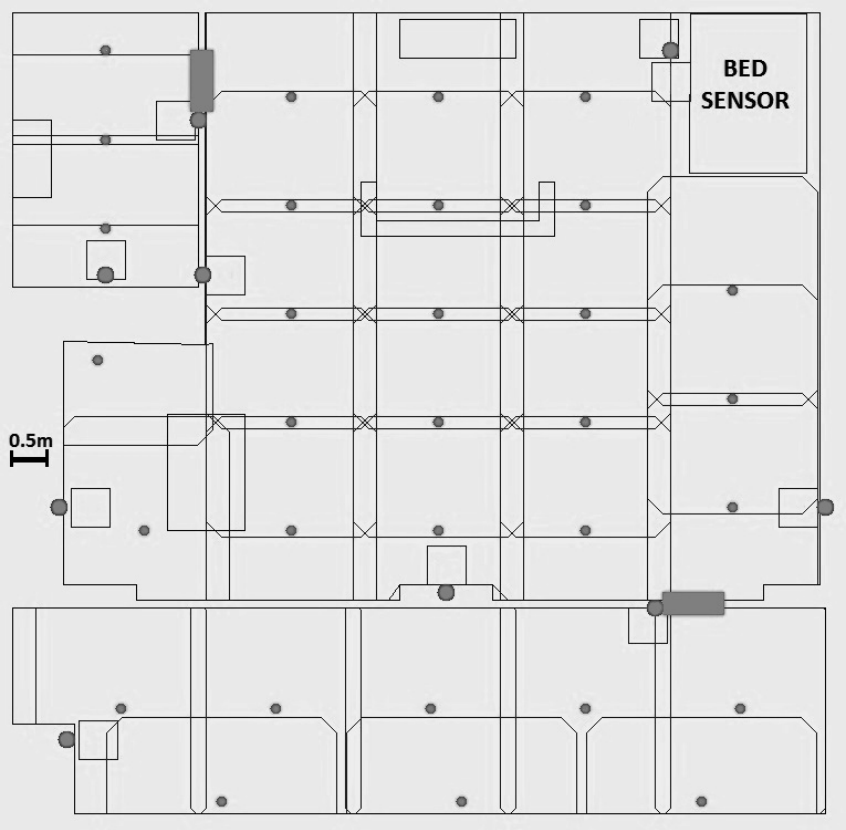

Figure 5.3: A fairly *dense* configuration of 32 binary sensors (31 motion sensors and a single occupancy sensor installed in the bed). All motion sensors are installed under the ceiling and vertically facing the floor. Small dots show the XY position of the motion sensors. The range of each sensor on the floor is expanded by $bodyR = 0.25m$ to support a much more realistic model of the person's body as discussed in section 4.6.1. Big dots show the XY position of nine switch-type sensors. Small squares next to switch-type sensors are their $bodyRange$ as introduced in section 4.2.1 and 4.5. These nine switch-type sensors are enabled in dns_{swt} and disabled in dns.

about $6cm$ (8%) of the mean error using the min sensor configuration, whereas this improvement for the dns configuration is just about $2.6cm$ which is almost 5%. Besides, according to the standard error in the plot, the difference between the two errors for the dns configuration is very close to be insignificant. This observation can be explained based on the characteristics of the pruning algorithm which improves the accuracy in sparser sensor configurations with larger coded regions. This claim, along with some relevant experiments, will be explained in the section about the impact of the pruning algorithm. Therefore, since the impact of the pruning algorithm in the dense configuration (dns) is weaker, the improvement added by the smooth movement – which is in fact because of the pruning algorithm – is lesser.

In the second part, we use a fixed sensor configuration: min. In this part we examine the effect of S and B, when using ideal (fn_0) or realistic (fn_1) noise models. The results are reported in the table 5.2. The only new observation here is that, the improvement added by the smooth movement with the realistic noise model ($10cm$ or 12.6%) is higher than that with the ideal noise model ($6cm$ or

Table 5.1: The mean localization error for fn_0 and ex_{on}. The first number in the parenthesis is the standard deviation and the second number is the standard error.

	Sensor Configuration	
Raw Trace	min	dns
S	0.666 m (0.293, 0.017)	0.485 m (0.277, 0.016)
B	0.725 m (0.293, 0.013)	0.511 m (0.25, 0.011)

Table 5.2: The mean localization error for min and ex_{on}. The first number in the parenthesis is the standard deviation and the second number is the standard error.

	Noise Model	
Raw Trace	fn_0	fn_1
S	0.666 m (0.293, 0.017)	0.692 m (0.298, 0.017)
B	0.725 m (0.293, 0.013)	0.790 m (0.291, 0.012)

8%). This difference in the amount of improvement is explained by the behavior of motion sensors in the realistic model. They are not triggered when the person is stationary. Hence, the bursty movement which has frequent stops will result in extra error when using a realistic noise model.

5.2 Sensor configuration

In this section, we analyze the correlation of the sensor configuration and the localization error. Throughout this section we fix the states of the noise model and the no-exclusion algorithm to fn_0 and ex_{on} respectively. In the first part, we rearrange the results from the previous section to compare the accuracy of localization using min and dns, under the conditions of S and B movements. Results are reported in the table 5.3. The straightforward observation here is that, when using dns with 32 binary sensors the mean error is lowered by around 28% compared to when using min with 21 binary sensors. In fact, the number of sensors has increased by 52%, so if the mean error was lowered by 34%, then the localization error would be in a reverse relation with the number of sensors. But this improvement is 28%, which shows the localization error is not improved as much as we have increased the sensor density (reaching, conceivably, a point of diminishing returns). This improvement is actually 27% with the smooth movement and 29% with the bursty movement. This little difference in the amount of improvement was analyzed in the previous section.

In the last part of this section, we observe the improvement added by switch-type sensors. We

Table 5.3: The mean localization error for fn_0 and ex_{on}. The first number in the parenthesis is the standard deviation and the second number is the standard error.

	Raw Trace	
Sensor Configuration	S	B
min	0.666 m (0.293, 0.017)	0.725 m (0.293, 0.013)
dns	0.485 m (0.277, 0.016)	0.511 m (0.25, 0.011)

Table 5.4: The mean localization error for fn_0, S, and ex_{on}. The first number in the parenthesis is the standard deviation and the second number is the standard error.

	Sensor Configuration	
Switches	min	dns
Off	0.666 m (0.293, 0.017)	0.485 m (0.277, 0.016)
On	0.571 m (0.31, 0.018)	0.415 m (0.262, 0.015)

Table 5.5: The mean localization error for min, fn_0, and ex_{on}. The first number in the parenthesis is the standard deviation and the second number is the standard error.

	Raw Trace	
Pruning Step	S	B
On	0.666 m (0.293, 0.017)	0.725 m (0.293, 0.013)
Off	0.882 m (0.306, 0.018)	0.893 m (0.308, 0.013)

fix the type of movement to S and compare the localization error using min vs. min_{swt} and dns vs. dns_{swt}. Results are reported in the table 5.4. The straightforward observation here is that the nine switch-type sensors improve the mean error by $9cm$ (14%) when using min. This improvement for dns is $7cm$ (14%).

5.3 Pruning step

In this section, we investigate the improvement added by the pruning step discussed in section 4.4.4. In the experiments, we fix the noise model and the state of the no-exclusion algorithm to fn_0 and ex_{on} respectively. Then for two sensor configurations – min and dns – we compare the error of the localization system with the enabled and disabled pruning step, for the smooth and bursty traces. Results are reported in tables 5.5 and 5.6. The straightforward observation here is that, as expected, the pruning algorithm significantly improves the mean error of localization. This improvement is $22cm$ (25%) when using min and with the smooth trace. It is $17cm$ (20%) when using min and B, $14cm$ (22%) when using dns and S, and finally $13cm$ (20%) when using dns and B. We can see that the amounts of improvement in experiments using the smooth trace are generally higher. This is because the pruning algorithm assumes a constant speed for the target person. So it naturally works better for a smooth trace rather than a bursty one. We can also see that, this improvement is slightly higher for min sensor configuration compared to dns. Although, this difference is really small and

Table 5.6: The mean localization error for dns, fn_0, and ex_{on}. The first number in the parenthesis is the standard deviation and the second number is the standard error.

	Raw Trace	
Pruning Step	S	B
On	0.485 m (0.277, 0.016)	0.511 m (0.25, 0.011)
Off	0.630 m (0.3, 0.017)	0.644 m (0.262, 0.011)

Table 5.7: The mean localization error for min and S. The first number in the parenthesis is the standard deviation and the second number is the standard error.

	Noise Model				
No-exclusion	fn_0	fn_1	fn_2	fn_3	fn_4
ex_{on}	0.666 m (0.293, 0.017)	0.692 m (0.298, 0.017)	0.719 m (0.333, 0.019)	0.799 m (0.444, 0.025)	0.914 m (0.47, 0.027)
ex_{off}	0.680 m (0.299, 0.017)	0.708 m (0.308, 0.017)	0.739 m (0.335, 0.019)	0.791 m (0.431, 0.025)	0.895 m (0.607, 0.035)

Table 5.8: The mean localization error for dns and S. The first number in the parenthesis is the standard deviation and the second number is the standard error.

	Noise Model				
No-exclusion	fn_0	fn_1	fn_2	fn_3	fn_4
ex_{on}	0.485 m (0.277, 0.016)	0.616 m (0.406, 0.023)	0.660 m (0.386, 0.022)	0.789 m (0.442, 0.025)	1.192 m (0.685, 0.039)
ex_{off}	0.606 m (0.297, 0.017)	0.676 m (0.357, 0.02)	0.705 m (0.315, 0.018)	0.770 m (0.414, 0.024)	0.986 m (0.727, 0.042)

may be even negligible in these experiments, there is an interesting analysis for why the pruning algorithm has a generally stronger impact for sparser sensor configurations. The pruning algorithm is practically extrapolating the person's position from the moment they enter a coded region to the moment they exit it and enter the next region. In the min-coverage configuration, the coded regions are larger and this extrapolation better improves the accuracy. But, in dense configurations, coded regions are smaller and the extrapolation done by the pruning algorithm still helps but not as much. For a more clear understanding, imagine an extreme case with an abnormally super-dense sensor configuration where coded regions are as small as $10cm * 10cm$ squares. Obviously in such a case, extrapolation is meaningless; so, the pruning algorithm wouldn't help at all.

Finally, we observe that using ideal binary sensors, the localization with a disabled pruning step has almost the same mean error for either of the movement types. It is simply because the only stage of the localization algorithm which takes into account a presumed value for the person's speed is the pruning stage. If we have noise-free and ideal binary sensors, without the pruning algorithm the system's performance is not impacted by the person's speed and style of movement.

5.4 Effect of noise and the no-exclusion algorithm

The no-exclusion algorithm – as discussed in section 4.6.6 – is aimed to handle the most challenging type of noise i.e. frequent false-negative signals and it therefore makes sense to test it jointly with the noise model. To do so, we have setup two sets of experiments: one with min and one with dns. In both sets, we fix the type of movement to S. In each set, for five different noise models we measure the localization error for ex_{on} and ex_{off}. Results are reported in tables 5.7 and 5.8. In the table 5.7, using the min-coverage configuration, the mean error of localization is increased as the probability

of false-negative signals increases in the noise models. This increase was obviously expected. We furthermore observe that using min, there is no significant difference in the mean error whether using ex_on or ex_off. The explanation is that the no-exclusion algorithm makes a difference in the result only if the sensor ranges have considerable overlaps. In fact, for a non-overlapping sensor configuration, the logical outcome of the localization algorithm with the enabled or disabled no-exclusion part is *exactly* the same. Our min configuration is not completely non-overlapping, and that explains the little difference in the mean error of ex_{on} and ex_{off} states; those small overlaps are not enough to make a significant difference in the mean error.

In the table 5.8, using the dense sensor configuration, we observe that the enabled no-exclusion algorithm increases the mean error for the first three noise models : fn_0, fn_1, and fn_2. Yet, the amount of this increase in error is reduced as the probability of false-negative signals increases in the noise models. In fn_3, for the first time the mean error for ex_{off} becomes lower than that for ex_{on}, but the difference is still insignificant. Finally, in the last noise model – fn_4 – which has a very probability of false-negative signals, the enabled no-exclusion method achieves a significantly better mean error than the default system.

Generally, for any sensor configuration there is a threshold on the frequency of false-negative signals. Before this threshold, the false-negative signals are not actually frequent enough to use the no-exclusion method. So, if we use the no-exclusion method before this threshold, we practically do not use important information in the zero signals which convey information that the person is not located in certain areas, and consequently we lose accuracy. Around this threshold, using or not using the no-exclusion method does not make a significant difference. Above this threshold, the frequency of false-negative signals is high enough that ignoring the information in zero signals improves the accuracy. This threshold has a reverse correlation with the amount of sensing range overlaps in a sensor configuration. For sensor configurations which use motion sensors with wide ranges where the range of a motion sensor may overlap with the range of many other sensors, the threshold can be very small. For such configurations, the no-exclusion method should be always enabled in real-world applications, in order to have a robust system.

Chapter 6

Conclusions and future work

In our indoor localization framework we used three types of sensors: (1) passive infrared motion sensors, (2) occupancy sensors for beds and chairs with fixed locations, and (3) switch-type sensors. We first elaborated the main part of our framework that only uses binary sensors and assumes they are ideal. We then integrated switch-type sensors. Finally, we enumerated five main challenges that we have encountered in a real environment with real sensors : (i) the real person's body model which has thickness and is not a single point, (ii) the oscillation of the positive signal of motion sensors, (iii) doors which are not guaranteed to be always closed or open, (iv) motion sensors which are not triggered when the person in their range is stationary, and (v) false-negative signals of motion sensors. We described how each of these challenges may cause problems with making invalid and/or misleading sensor readings. We explained our solutions to address each problem.

In experiments, we focused on five dimensions of variability: (a) the type of the person's movement, i.e. smooth vs. bursty movement, (b) the sensors' configuration, i.e. minimum-coverage vs. dense, and enabled vs. disabled switch-type sensors, (c) enabled vs. disabled pruning step, (d) enabled vs. disabled no-exclusion approach, and (e) the noise model, i.e. ideal vs. realistic vs. three heavily noisy models. In summary, using ideal sensors, we have achieved $67cm$ and $49cm$ of mean localization error with a minimum-coverage and a fairly dense sensor configurations respectively. For a realistic model of sensors, these numbers are $69cm$ and $62cm$ respectively. We have drawn conclusions from two important observations of the experiments.

Experiments show that in the case of ideal sensors, the localization error is lowered by 28% when the sensor density was raised from 0.175 to 0.267 binary sensors per squared meters. Hence, in terms of accuracy – i.e. $1/error$ – the localization accuracy has improved by 39% with a 52% increase in the sensor density. Therefore, the localization accuracy has not improved as much as the sensor density has increased. It should be emphasized that these numbers were for the case of ideal sensors, whereas with a realistic noise model (fn_1) the situation is very different. With 52% increase in the density of real sensors, only 13% improvement in the accuracy is obtained. This is because as the sensor density increases, the overlaps of sensor ranges increase too. As a result, the negative impact of false-negative signals in a realistic model is magnified with more range overlaps.

This partially cancels the improvement added by the increase in the sensor density. In summary, using real binary sensors, if we incur additional cost to place more sensors of this type, we cannot expect to gain proportional degree accuracy.

The last set of experiments demonstrated that with the minimum-coverage sensor configuration, the no-exclusion method almost does not change the localization accuracy. This is because in the min-coverage sensor configuration, sensor ranges have little overlaps. Therefore, for such a sensor configuration the logic of no-exclusion algorithm has almost the same outcome as the default setting (without no-exclusion). But, with the dense sensor configuration, the no-exclusion algorithm worsens the accuracy for fn_0, fn_1, and fn_2 noise models. For fn_4 it slightly improves and for fn_5 it remarkably improves the mean error of localization. We conclude that even using a fairly dense configuration of narrow-range motion sensors, the no-exclusion method only improves the localization accuracy for very noisy models. So, by default and for a normal realistic noise model, we should certainly disable the no-exclusion method.

There are three important future directions pertaining to the results of this thesis: (a) an automatic optimal sensor placement, (b) employing wide-range motion sensors, and (c) combining the information about the geometry of sensor coverage with a traditional state estimation framework. We can see in the experiments that the sensor configuration contributes significantly to the performance of our localization framework. Currently, the process of sensor placement is completely manual. This makes it very time consuming. Besides, for a given number of sensors, the result of a manual sensor placement are arguably non-optimal. An *optimal* sensor placement of a fixed number of sensors is the one that minimizes the expected mean error of localization. Due to the above-mentioned reasons, it is crucial to design and implement an automatic sensor placement system. This system should be able to take as input the building model and the number of sensors of each type, and output an optimal sensor configuration.

The second work that should be done in the future is to add wide-range motion sensors. Currently, we only use the narrow-range motion sensors. This makes the system more robust to false-negative signals and other sources of noise. By using wide-range motion sensors, we can obtain fine grained coded regions with significantly smaller number of sensors. On the other hand, we observed in the experiments that the no-exclusion algorithm is not effective when using sensor configurations with little overlaps. With wide-range motion sensors, the no-exclusion algorithm could be more effective. A good future strategy would be to employ wide-range sensors and develop the no-exclusion algorithm to still maintain robustness against false-negative signals.

The presented work is heavily influenced by the attention to the geometry of the sensor coverage. In the future, it would be interesting to consider how information about the geometry of sensor coverage can be combined with a traditional state estimation framework, using e.g., state estimation techniques applicable to binary sensors. This future work will also help assess the degree to which accuracy of geometric coverage representation has a significant impact on the state estimation

outcomes.

Bibliography

[1] Olivier Barnich and Marc Van Droogenbroeck. Droogenbroeck, vibe: a powerful random technique to estimate the background in video sequences. In *in International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2009*, pages 945–948, 2009.

[2] Richard J. Barton and Divya Rao. Performance capabilities of long-range uwb-ir tdoa localization systems. *EURASIP J. Adv. Signal Process*, 2008:81:1–81:17, January 2008.

[3] Esteban Tobias Bayro Kaiser. Indoor simultaneous localization and mapping for pedestrian with wearable computing. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, MobileHCI '10, pages 487–488, New York, NY, USA, 2010. ACM.

[4] Mark d. Berg, Otfried Cheong, Marc Kreveld, and Mark Overmars. *Computational Geometry*. Springer Berlin Heidelberg, 2008.

[5] M. Bertozzi, A. Broggi, C. Caraffi, M. Del Rose, M. Felisa, and G. Vezzoni. Pedestrian detection by means of far-infrared stereo vision. *Comput. Vis. Image Underst.*, 106(2-3):194–204, May 2007.

[6] N. M. Boers, D. Chodos, P. Gburzynski, L. Guirguis, J. Huang, R. Lederer, L. Liu, I. Nikolaidis, and E. Stroulia. The smart condo project: Services for independent living. *Smart Healthcare Applications and Services*, 2011.

[7] Mustapha Boushaba, Abdelhakim Hafid, and Abderrahim Benslimane. High accuracy localization method using aoa in sensor networks. *Comput. Netw.*, 53(18):3076–3088, December 2009.

[8] Hongyang Chen, Bin Liu, Pei Huang, Junli Liang, and Yu Gu. Mobility-assisted node localization based on toa measurements without time synchronization in wireless sensor networks. *Mob. Netw. Appl.*, 17(1):90–99, February 2012.

[9] J. Cobos, L. Pacheco, X. Cufi, and D. Caballero. Integrating visual odometry and dead-reckoning for robot localization and obstacle detection. In *Proceedings of the 2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR) - Volume 01*, AQTR '10, pages 1–6, Washington, DC, USA, 2010. IEEE Computer Society.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[11] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886 –893 vol. 1, june 2005.

[12] E. Foxlin. Pedestrian tracking with shoe-mounted inertial sensors. *Computer Graphics and Applications, IEEE*, 25(6):38 –46, nov.-dec. 2005.

[13] Kazutaka Fukuda and Eiji Okamoto. Performance improvement of toa localization using imr-based nlos detection in sensor networks. In *Proceedings of the The International Conference on Information Network 2012*, ICOIN '12, pages 13–18, Washington, DC, USA, 2012. IEEE Computer Society.

[14] Michael Harville and Dalong Li. Fast, integrated person tracking and activity recognition with plan-view templates from a single stereo camera. In *IN: IEEE CONF. ON COMPUTER VISION AND PATTERN RECOGNITION*, pages 398–405, 2004.

[15] Andreas Hub, Joachim Diepstraten, and Thomas Ertl. Design and development of an indoor navigation and object identification system for the blind. *SIGACCESS Access. Comput.*, (77-78):147–152, September 2003.

[16] Shahram Jalaliniya and Thomas Pederson. A wearable kids' health monitoring system on smartphone. In *Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design*, NordiCHI '12, pages 791–792, New York, NY, USA, 2012. ACM.

[17] Omar Javed, Khurram Shafique, and Mubarak Shah. A hierarchical approach to robust background subtraction using color and gradient information. In *Proceedings of the Workshop on Motion and Video Computing*, MOTION '02, pages 22–, Washington, DC, USA, 2002. IEEE Computer Society.

[18] Wooyoung Kim, Kirill Mechitov, Jeung-Yoon Choi, and Soo Ham. On target tracking with binary proximity sensors. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, IPSN '05, Piscataway, NJ, USA, 2005. IEEE Press.

[19] Mikkel Baun Kjærgaard, Henrik Blunck, Torben Godsk, Thomas Toftkjær, Dan Lund Christensen, and Kaj Grønbæk. Indoor positioning using gps revisited. In *Proceedings of the 8th international conference on Pervasive Computing*, Pervasive'10, pages 38–56, Berlin, Heidelberg, 2010. Springer-Verlag.

[20] T. Klingeberg and M. Schilling. Mobile wearable device for long term monitoring of vital signs. *Comput. Methods Prog. Biomed.*, 106(2):89–96, May 2012.

[21] Yang Sun Lee, Ji-Min Lee, Sang Soo Yeo, Jong Hyuk Park, and Leonard Barolli. A study on the performance of wireless localization system based on aoa in wsn environment. In *Proceedings of the 2011 Third International Conference on Intelligent Networking and Collaborative Systems*, INCOS '11, pages 184–187, Washington, DC, USA, 2011. IEEE Computer Society.

[22] Chin-Heng Lim, Yahong Wan, Boon-Poh Ng, and C. M.S. See. A real-time indoor wifi localization system utilizing smart antennas. *IEEE Trans. on Consum. Electron.*, 53(2):618–622, May 2007.

[23] Felix Mata, Andres Jaramillo, and Christophe Claramunt. A mobile navigation and orientation system for blind users in a metrobus environment. In *Proceedings of the 10th international conference on Web and wireless geographical information systems*, W2GIS'11, pages 94–108, Berlin, Heidelberg, 2011. Springer-Verlag.

[24] T. Murakita, T. Ikeda, and H. Ishiguro. Human tracking using floor sensors based on the markov chain monte carlo method. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 4, pages 917 – 920 Vol.4, aug. 2004.

[25] Wayne Chelliah Naidoo and Jules-Raymond Tapamo. A model of an intelligent video-based security surveillance system for general indoor/outdoor environments. In *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*, SAICSIT '08, pages 159–168, New York, NY, USA, 2008. ACM.

[26] T. Nishizeki and N. Chiba. *Introduction to Algorithms*. Elsevier Science Publishers, 1988.

[27] S. Noimanee, T. Tunkasiri, K. Siriwitayakorn, and J. Tuntrakoon. Wireless c45 based vital-signs monitoring system for patient after heart operation care. In *Proceedings of the 7th WSEAS International Conference on Electronics, Hardware, Wireless and Optical Communications*, EHAC'08, pages 239–243, Stevens Point, Wisconsin, USA, 2008. World Scientific and Engineering Academy and Society (WSEAS).

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

[29] Joseph O'Rourke. *Algorithmic Geometry*. Cambridge University Press, 1998.

[30] Robert J. Orr and Gregory D. Abowd. The smart floor: a mechanism for natural user identification and tracking. In *CHI '00 extended abstracts on Human factors in computing systems*, CHI EA '00, pages 275–276, New York, NY, USA, 2000. ACM.

[31] Veljo Otsason, Alex Varshavsky, Anthony LaMarca, and Eyal de Lara. Accurate gsm indoor localization. In *Proceedings of the 7th international conference on Ubiquitous Computing*, UbiComp'05, pages 141–158, Berlin, Heidelberg, 2005. Springer-Verlag.

[32] R. F. Pinkston. A touch sensitive dance floor/MIDI controller. *Acoustical Society of America Journal*, 96:3302, November 1994.

[33] Franco P. Preparata and Michael I. Shamos. *Computational geometry: an introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[34] N. Shrivastava, R. Mudumbai U. Madhow, and S. Suri. Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 251–264, New York, NY, USA, 2006. ACM.

[35] Thiago Teixeira, Gershon Dublon, and Andreas Savvides. A survery of human-sensing: Methods for detecting presence, count, location, track, and identity. *ACM Computing Surveys*, 2010.

[36] Jorge Torres-Solis and Tom Chau. Wearable indoor pedestrian dead reckoning system. *Pervasive Mob. Comput.*, 6(3):351–361, June 2010.

[37] M.A. Turk and A.P. Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on*, pages 586 –591, jun 1991.

[38] Alex Varshavsky, Eyal de Lara, Jeffrey Hightower, Anthony LaMarca, and Veljo Otsason. Gsm indoor localization. *Pervasive Mob. Comput.*, 3(6):698–720, December 2007.

[39] Harshvardhan Vathsangam, Anupam Tulsyan, and Gaurav S. Sukhatme. A data-driven movement model for single cellphone-based indoor positioning. In *Proceedings of the 2011 International Conference on Body Sensor Networks*, BSN '11, pages 174–179, Washington, DC, USA, 2011. IEEE Computer Society.

[40] Paul Viola and Michael Jones. Robust real-time object detection. In *International Journal of Computer Vision*, 2001.

[41] Zijian Wang, Eyuphan Bulut, and Boleslaw K. Szymanski. Distributed energy-efficient target tracking with binary sensor networks. *ACM Trans. Sen. Netw.*, 6(4):32:1–32:32, July 2010.

[42] Tracy L. Westeyn. *Child's play: activity recognition for monitoring children's developmental progress with augmented toys*. PhD thesis, Atlanta, GA, USA, 2010. AAI3425169.

[43] Le Yang and K. C. Ho. An approximately efficient tdoa localization algorithm in closed-form for locating multiple disjoint sources with erroneous sensor positions. *Trans. Sig. Proc.*, 57(12):4598–4615, December 2009.