ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Engineering Angle-of-Arrival-based Indoor Localization Systems via Simulation

Tesi di laurea in
Sistemi Autonomi

Relatore
Prof. Andrea Omicini

Candidato
Shapour Nemati

Correlatore
Dott. Giovanni Ciatto
Dott. Salvador Santonja

Quarta Sessione di Laurea
Anno Accademico 2019-2020

Abstract

Indoor localization is a hot research topic yet to find a shared agreement on the key methods and technologies enabling a satisfactory solution to the problem. Among the plethora of techniques currently being studied, "Angle of Arrival" (AoA) has recently gained momentum as the Bluetooth Special Interest Group introduced it in the new Bluetooth 5.1 standard as the release's major enhancement.

In Bluetooth-based AoA, a set of locator nodes are able to compute the angle between themselves and a to-be-located device sending an ad-hoc crafted Bluetooth packet. Given the angles and the positions of the locators, a data fusion algorithm can compute the approximate relative position of the observed device.

In the right conditions, AoA can reach sub-meter accuracy. However only the lowest levels of the stack are already provided, and the implementation of the data fusion layer is left to the developers. Correctly engineering this layer is not an easy task, as it would require a complex tuning process through trial and error, involving cumbersome real world tests.

The goal of this thesis is to build a tool aimed at supporting the engineering of AoA-based localization applications, both *in-silico* and in practice. Along this line, we develop an actor-based software framework that *(i)* is agnostic with respect to the specific technology used to realize AoA, and *(ii)* can work with both simulated and real AoA-based sensors.

The proposed framework is developed in the context of the European project "PRYSTINE", and aims at achieving a full-stack localization system based on Bluetooth-based AoA as the physical medium, and *particle filtering* as the data-fusion algorithm. The proposed framework, however, is general enough to support any other technology capable of calculating AoA, and any other filtering algorithm for data fusion.

A mio padre,
che scherzando riesce a prendere tutto con filosofia

Acknowledgements

Throughout my master thesis, I have received much support from family, friends, and University staff.

First, thanks to everyone who helped me prepare my thesis, starting from the researchers of ITI and UPV for trusting me in taking important decisions for their projects. Thanks to my supervisor Andrea Omicini, for granting me the freedom over the choice of the thesis' matter. Thanks to my co-supervisor Giovanni Ciatto, for always shedding light on the thesis, leading me to the correct path.

Thanks to my friends, for being there in all moments of life, important and ordinary alike, both when we were distant and when we were close. Thanks to my family for always supporting me, respecting my time and choices, and cheering my achievements.

Contents

Abstract **iii**

1 Introduction **1**

2 State of the Art **5**
2.1 Indoor Localization 5
2.1.1 Trilateration 6
2.1.2 Triangulation 7
2.2 Data Fusion 8
2.2.1 Hidden Markov Models 11
2.2.2 Kalman Filters 11
2.2.3 Dynamic Bayesian Networks 12
2.3 Bluetooth 5.1 localization 13
2.3.1 Angle of Arrival and Angle of Departure 13
2.3.2 Technical details 15
2.3.3 Performance 16

3 Design **17**
3.1 Requirements Analysis 17
3.2 Problem Analysis 18
3.2.1 AoA and data fusion 18
3.2.2 Heterogeneous distributed system 20
3.3 Architecture 20
3.3.1 Structure 21
3.3.2 Interaction 21
3.3.3 Behavior 23

4 Implementation **27**
4.1 Actor System 28
4.2 Particle Filtering 31
4.3 Simulation 33

4.4 Situated System 34

5 Validation **37**
5.1 Experimental Results 37
5.1.1 Simulation 38
5.1.2 Physical Experiments 42
5.2 Code Quality 45
5.2.1 Static Analysis 45
5.2.2 Testing 46

6 Conclusions **47**
6.1 Future works 48

List of Figures

2.1 A simple example of trilateration, where point A is the ObservedItem, while points B, C, and D are Locators. 8
2.2 Angle of arrival method, with a multi-antenna array on the receiver. Source: [10] 14
2.3 Angle of departure method, with a multi-antenna array on the transmitter. Source: [10] 14
2.4 How trigonometry is used to compute angle of arrival. Source: [10] 15

3.1 A component diagram showing the overall system structure and its main actors. 22
3.2 The sequence diagram describing the overall interaction flow. 24
3.3 The state diagram, divided in three lanes. The leftmost lane contains the ObservedItem behavior, the central one describes the Locator, and the rightmost is for the LocalizationService. . . 24

4.1 The sequence diagram describing the overall message exchange revolving around the "step" and "stop" messages for coordination. . . 29
4.2 The state diagram of the ObservedItem. "LLD" is short for low level data. 30
4.3 The state diagram of the Locator. 31
4.4 The state diagram of the LocalizationService. 32
4.5 An architectural overview of the situated system. The "transmitter" is the ObservedItem, the "receivers" are the Locators. 35

5.1 Visual representation of the particle filtering applied to simulated data where the ObservedItem moves at constant speed. 39
5.2 Visual representation of the particle filtering applied to simulated data where the ObservedItem moves at constant speed and then stops. 39

5.3 Average distances for different parameters values, with the ObservedItem moving at constant speed. The four graphs each represent scenarios with a different number of Locators: cyan has 2, red has 3, blue has 4, and green has 5. Each line in each graph represents the number of particles employed: the solid line represents 10 particles, the dashed one 50, and the dash-dot one 100, and the dotted one 500. 40
5.4 Average distances for different parameters values, with the ObservedItem moving at constant speed for the first half, and standing still for the second half. The four graphs each represent scenarios with a different number of Locators: cyan has 2, red has 3, blue has 4, and green has 5. Each line in each graph represents the number of particles employed: the solid line represents 10 particles, the dashed one 50, and the dash-dot one 100, and the dotted one 500. . 41
5.5 Position of the two Locators (A and B), and the points through which the Locator passes through in the different scenarios (C, D, E, and F). 42
5.6 Average distances for different parameters values, using real data in which the ObservedItem moves at a constant speed. Each line in the graph represents the number of particles employed: the solid line represents 10 particles, the dashed one 50, and the dash-dot one 100, and the dotted one 500. 43
5.7 Average distances for different parameters values, using real data in which the ObservedItem stands still for the whole duration. Each line in the graph represents the number of particles employed: the solid line represents 10 particles, the dashed one 50, and the dash-dot one 100, and the dotted one 500. 44
5.8 Visual representation of the particle filtering applied to real world data where the ObservedItem stands still. 44
5.9 Visual representation of the particle filtering applied to real world data where the ObservedItem moves along two lines. 44

Listings

5.1 Setup of parameters in the framework for fine tuning 38

Chapter 1

Introduction

The recent advancements in computing are making situated systems increasingly frequent [11]. Be it IoT [6], Smart Cities [4], or robotics [9], nowadays many complex systems rely on location-awareness for providing advanced functionalities. While some techniques such as GPS are widespread, standardized, and perform well [2], they also have a major downside: they do not work or do not provide accurate measurements in indoor environments.

The need for indoor localization opened new research paths as many fields are deeply entangled with situatedness [11]. A variety of techniques and technologies has been proposed through the years, none of them really solving the problem in a satisfactory manner, as many issues such as accuracy, energy efficiency, cost, range, and scalability are still problematic.

Recently, the Bluetooth Special Interest Group (SIG) issued a new version of Bluetooth: the 5.1 specification, which comes with a localization functionality based on *Angle-of-Arrival* (AoA) [8]. This technique is not new but never found a fertile field in any specific technologies that could enable its usage in modern contexts. Bluetooth is a promising technology that could actually bring this new technique to become a standard, as Bluetooth itself is already widespread, and little hardware modification is required.

AoA is a good fit for situated systems as it does not require complex new topologies and additional nodes, instead, it just blends in easily at the only cost of adding antenna arrays to some nodes. Furthermore, triangulation-based systems scale well with the size of the system: just two locator nodes are enough for a 2D localization, but increasing their number would provide more data that can be used for better estimating the true position of the observed object.

Among the most promising results, some studies on the technique [5] show that it is possible to achieve sub-meter accuracy using AoA and specialized hardware. However, these studies are made in perfect conditions, such as anechoic chambers, or using expensive hardware, whereas real-world scenarios need to deal with in-

terferences, cheap hardware, and many other problems. This kind of issue has already been addressed multiple times when dealing with other problems, so the scientific literature already has much information on methods made for dealing with uncertainty [2].

While these methods have strong theoretical foundations, they need to fit the specific use case by tuning complex sets of parameters. This activity is rather time-consuming and error-prone, on top of being deeply related to the specific system at hand, which means solving the problem once does not imply the same solution can be used in a different setting.

The goal of this thesis is thus to provide a tool that could aid professionals in the process of correctly engineering their system so that it performs as requested, without the need for repetitive and expensive experiments in a real-world scenario. The framework has been developed in a context where the technology of reference was the novel Bluetooth 5.1 standard but *does not rely on it as an essential part*, instead, it can leverage any technology providing an angle between the locator and the observed object. The same goes for the data fusion algorithm: particle filtering has been chosen for the specific case, but any other algorithm or methodology could be implemented and used in this scenario.

Data fusion is essential for this kind of system, because the sensor's measurements can be noisy, and without employing any additional technique, the end result would yield an unstable sequence of positions. In particular, filtering is essential as it is the technique that enables the data fusion to be applied in real time, estimating the current position of an object base on the information available up to that moment of time.

One of the most important aspects of the system is the possibility of performing simulations, which just need the sensor's noise's statistical model to get started with simulating with the end of fine-tuning the system even before it is actually built. When the system is in use and some data are available, it is possible to record the data and use it later as an input to the framework in order to run the fusion algorithm with different settings, finding the parameters which best fit the real-world case.

To prove the effectiveness of the approach, we discuss a scenario where a few devices collect real world data to feed into the system. The use case is based on Bluetooth 5.1's new specifications, powered by Texas Instruments' development boards, and three simple movement patterns.

Thesis Structure. The remainder of this thesis is structured as follows. Chapter 2 discusses the different localization techniques that can be used in an indoor environment, makes an overview of the possible models and algorithms for data fusion, and provides an explanation of the novel Bluetooth 5.1 standard's local-

ization capabilities. Chapter 3 describes the rigorous approach that takes the system's requirements and turns them into an architecture that could satisfy the specified needs. Chapter 4 comprises of some lower-level details that are still interesting for understanding how the system was built and what are the reasons behind their decision-making. In particular, Actor-paradigm, python programming language, particle filtering, and simulation, are the major topics addressed. Chapter 5 explains the metrics used for understanding if the system was successful or not in delivering the expected outcome. More information on the case study is provided here. Finally, Chapter 6 concludes this thesis by summarizing its main contributions and envisioning possible future works.

Chapter 2

State of the Art

This chapter discusses the theoretical concepts that are fundamental for indoor localization, data fusion, and Bluetooth-based localization. Section 2.1 provides information on indoor localization, why it is different from outdoor localization, and the two main techniques used: trilateration and triangulation. Section 2.2 explains the mathematical and algorithmic approaches in dealing with uncertainty and multiple sources of data, describing the main methods that could apply to indoor localization. Finally, Section 2.3 briefly describes the novel Bluetooth (BT) 5.1 specification that enables localization services to be implemented on top of the BT stack.

In what follows we denote relevant, domain-specific terms in small caps. Such terms are then further analysed and discussed in the remainder of this thesis, because of their pivotal role.

2.1 Indoor Localization

We define Localization as a process where one ore more Observed items are recognized, and their coordinates in a given reference system is *continuously* provided by means of a disembodied Localization Service that aggregates the data of multiple Locators. Each Locator is a situated node capable of acquiring partial information about the location of one or more Observed items, and send it to the Localization Service for data fusion.

Indoor localization and outdoor localization might seem very similar problems at first glance, as they both fulfill the definition above, but a deeper study of the two shows very different scenarios. Just to name a few, indoor positioning usually relies on low to medium range communication technologies, needs to establish its reference system, and works in environments where the satellite signals could be inaccurate or not work at all.

The definition above mentions coordinates, which means we are more specifically speaking about a *positioning system*. This is an important remark, as among the most successful indoor localization techniques many of them are based on *proximity detection* [12]. In this case, the item to be located is not tracked continuously, and its position is not given only as coordinates. Instead, some locators called beacons act as reference points: if the item can exchange messages with them, then they are within communication range. If we consider a 2D scenario, this means the item could be located anywhere inside the area of the circle which has its center in the beacon and the communication range as its radius.

The proximity solution can be turned into a full positioning system employing additional measurements and multiple beacons, as explained in Section 2.1.1. However, things are not as easy as they seem, because the sensing layer always introduces some considerable noise, which leaves us with circles that do not have the right radius, causing the intersection points to either be 0 or more than 1. This problem is addressed in Section 2.2, here we work with the theoretically perfect, mathematical models.

There is a variety of enabling technologies, ranging from communication-based ones, to image analysis. The different technologies can provide different performance and trade-offs in cost, efficiency, accuracy, and many other metrics, but the fundamental techniques are completely independent of them. Whether the angle-of-arrival is calculated by sampling radio frequencies with different antennas or obtained employing computer vision algorithms based on a video stream, the same algorithms can be applied, possibly tuning the parameters that help to cope with the different noises caused by the underlying technology.

In the next subsections, we discuss the two major positioning techniques: trilateration and triangulation.

2.1.1 Trilateration

Trilateration is a technique used for localization where multiple Locators that are stationary and at known locations compute their distance from the item, so that the Localization Service can use this information to compute the position of the Observed Item.

The working principle of trilateration is easily understood when explained in the two-dimensional case, referencing basic geometrical concepts that still apply in 3D. We can first think of a single Locator and a scenario similar to the proximity detection, but with the additional information of the distance between the beacon and the item. Knowing a point and a distance we also know that the Observed Item's location belongs to the circumference that has the Locator as its center, and the distance as the radius.

We now consider two circles on the plane and their possible intersections.

- **No intersection** if their centers have a distance greater than the sum of their radii;

- **One point** if their centers have a distance exactly equal to the sum of their radii;

- **Two points** if their centers have a distance greater than 0 but lower than the sum of their radii;

- **Infinite points** if the two centers are the same.

In the trilateration technique, the first option can be discarded as we only consider beacons that have an item within range, and the last one can be discarded as two beacons will not be placed in the same spot. We are left with the option of one point, which is only true for exactly one point of the space, and thus improbable, and the one with two points, which is the most likely situation. Given the ambiguity of the two possible points, a different point of view is needed to solve the issue. This is done with an additional Locator that will provide other sets of points, and the intersection of those with the set mentioned above is one point: the location of the item. The same concepts apply in 3D, just with a bit more technicalities.

If we now imagine having multiple beacons scattered around an internal environment, making sure their "circles" overlap, it is possible to use trilateration to find the coordinates of the item, which turns the proximity solution into a fully-flagged positioning system, as shown in Figure 2.1.

2.1.2 Triangulation

Triangulation is a localization technique that only uses angles from fixed (and known) locations to the ObservedItem to determine its location. In a two-dimensional scenario, a point could theoretically be found by intersecting two non-parallel lines. In three dimensions, it is still possible to find the intersection point with just two lines, which need 2 angles each: one for the orientation on the 2D plane, and the other for its elevation angle.

However, real measurements are affected by noise. In a 2D scenario, if we had 3 Locators and noisy measurements, it would be very likely that the three lines do not meet in the exact point, rather each couple of lines would meet at a different point, leaving us with the ambiguity of knowing which point is the correct one. This problem is tackled with the usage of data fusion, as described in Section 2.2.

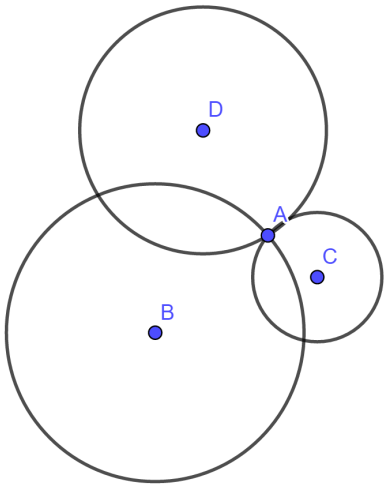

Figure 2.1: A simple example of trilateration, where point A is the ObservedItem, while points B, C, and D are Locators.

2.2 Data Fusion

Sensors used in situated systems make use of physical properties to make measurements that are digitally encoded and saved in some memory or directly sent to the device. The theoretical models based on ideal working conditions, the manufacture of the sensor, and the quantization of information, are just some of the factors that make the measurements noisy. A single sensor read could yield a result that greatly differs from the real world conditions, however, multiple measurements can be used to find the *most likely* world state.

Data fusion is the discipline employing different techniques in order to make use of different data (possibly from different sources and in different moments of time) to infer the most accurate world state based on the given measurements and some background knowledge. It makes use of well-studied mathematical and statistical concepts, theories, and models to approach the problem [7].

A very strong model is the Bayesian network, a directed acyclic graph where each node represents a random variable, and each edge a conditional probability. The topology of such a graph is based on cause-effect relationships in the domain, where a node has influence on another node if there is an edge connecting them. This simple model is the theoretical foundation of the next ones discussed in the rest of the section.

Before proceeding, it is important to recall the Bayes' rule, which states the

relation between a conditional probability and its parts:

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)}. \tag{2.1}$$

If some background evidence e is known, it can be used in the formula as follows:

$$P(Y \mid X, e) = \frac{P(X \mid Y, e)P(Y \mid e)}{P(X \mid e)}. \tag{2.2}$$

With the given premises, we are ready to discuss a specific kind of data fusion that addresses time and uncertainty, studying dynamic systems and their state over time. It is thus necessary to model such dynamic situations in a formal way.

Modeling dynamic situations. The model comprises of a series of time slices characterized by a set of random variables. Some of these variables, E_t, are observable, while others, X_t, are not. An observation at time t is expressed as the random variable taking a concrete value: $E_t = e_t$. Since this is a dynamic system, some models describing its evolution are needed: in particular the transition model and the sensor model are defined.

The transition model is the probability of the random variable X at time t, given the previous values of X: $P(X_t \mid X_{0:t-1})$. For the sake of simplicity we can employ the Markov assumption writing this as a first-order Markov process: $P(X_t \mid X_{0:t-1}) = P(X_t \mid X_{t-1})$. This means that the current state only depends on the previous one. Another assumption we can make is that the process is stationary, which means the state *does* change over time, but following the same laws at any given time. These assumptions just make the explanations easier, and the formulas more clear, but without them all of the following discussion still holds.

The sensor model (also called observation model) is the probability of the sensor read given the current state variable: $P(E_t \mid X_t)$.

Finally, the the prior probability distribution at time 0, $P(X_0)$ needs to be specified. This represents the initial conditions of the system.

Inference in temporal models. With the previously described model it is possible to perform many different inference tasks, such as prediction, smoothing, most likely explanation, learning, and filtering. We are interested in the last one.

Filtering computes a state estimation given the previous evidence. This is done by starting with an initial belief state, and updating it every time a new measurement is available instead of calculating everything from scratch. This process is also known as recursive estimation, and can be expressed with the formula below, for a function f.

$$P(X_{t+1} \mid e_{1:t+1}) = f(e_{t+1}, P(X_t \mid e_{1:t}))$$

This can be seen as a two-part process: first, the transition model is applied to the current belief state, then it is updated through the new sensor data, using the sensor model. The algorithm details depend on the f function, which is specified by the concrete model. We can rearrange the formula to explicitly state the two steps. We first expand the evidence in two parts.

$$P(X_{t+1} \mid e_{1:t+1}) = P(X_{t+1} \mid e_{1:t}, e_{t+1})$$

Then, we use Bayes' Rule Equation (2.2).

$$P(X_{t+1} \mid e_{1:t}, e_{t+1}) = \alpha P(e_{t+1} \mid X_{t+1}, e_{1:t}) P(X_{t+1} \mid e_{1:t})$$

Here, α is a normalizing constant that makes probabilities sum up to 1. Finally, we apply the sensor Markov assumption.

$$\alpha P(e_{t+1} \mid X_{t+1}, e_{1:t}) P(X_{t+1} \mid e_{1:t}) = \alpha P(e_{t+1} \mid X_{t+1}) P(X_{t+1} \mid e_{1:t}) \tag{2.3}$$

In Equation (2.3), the first term $P(e_{t+1} \mid X_{t+1})$ represents the sensor model, and the second one $P(X_{t+1} \mid e_{1:t})$ the transition model.

It is possible to rewrite this as a one-step prediction formula, which can be later used for recursive computation. We substitute the second term of the equation (the one representing the transition model) with a summation that is conditioned on the current state X_t.

$$P(X_{t+1} \mid e_{1:t+1}) = \alpha P(e_{t+1} \mid X_{t+1}) \sum_{x_t} P(X_{t+1} \mid x_t, e_{1:t}) P(x_t \mid e_{1:t})$$

Again, the Markov assumption can be employed for simplifying the formula.

$$P(X_{t+1} \mid e_{1:t+1}) = \alpha P(e_{t+1} \mid X_{t+1}) \sum_{x_t} P(X_{t+1} \mid x_t) P(x_t \mid e_{1:t}) \tag{2.4}$$

The $P(X_t \mid e_{1:t})$ can be seen as a "message" $f_{1:t}$ propagated along the sequence, modified by every transition and observation.

$$f_{1:t+1} = \alpha \textsc{Forward}(f_{1:t}, e_{t+1})$$

where FORWARD is the update described in Equation (2.4), and the initial "message" is $f_{1:0} = P(X_0)$.

In the following subsections we are going to describe three of the most important concrete models.

2.2.1 Hidden Markov Models

Hidden Markov Models have the peculiarity of using a *single* discrete random variable, which enables an elegant algorithmic solution. We start by defining the transition and sensor models in terms of the state variable X_t and the evidence variable's concrete value e_t.

Transition model: a matrix T of dimensions $S \times S$, where S is the number of different values that X_t can assume.

$$T_{ij} = P(X_t = j \mid X_{t-1} = i).$$

It should be read as: T_{ij} is the probability of transitioning from state i to state j.

Sensor model: a diagonal matrix O_t, of dimensions $S \times S$, whose $i$ths elements on the diagonal are $P(e_t \mid X_t = i)$, and everywhere else the value is 0.

With the given information, it is possible to write Equation (2.4) as

$$f_{1:t+1} = \alpha O_{t+1} T^\top f_{1:t}$$

.

The single variable is not a major impediment, as it is possible to use tuples as the random variable's values. However, the fact that it is discrete does make it unsuitable for many applications.

2.2.2 Kalman Filters

Kalman filters solve the problem of the discrete variable by treating the uncertainty in the problem as a Gaussian Distribution. The key idea is directly applying the transition model as a mathematical formula, and then adding some Gaussian noise to account for external factors that are not directly addressed by the mathematical model. The same goes for the **sensor model**, which is assumed to be affected by gaussian noise as well.

From a practical point of view, since Gaussian distributions keep their properties after the operations used in Bayesian networks, we can just update the distribution after each step obtaining another Gaussian distribution, so we can keep on applying in the same way. The prediction step can be written as

$$P(X_t \mid e_{1:t}) = \int_{x_t} P(X_{t+1} \mid x_t) P(x_t \mid e_{1:t}) dx_t$$

The updated distribution is

$$P(X_{t+1} \mid e_{1:t+1}) = \alpha P(e_{t+1} \mid X_{t+1}) P(X_{t+1} \mid e_{1:t})$$

Each step and each message of the FORWARD operator, are then characterized by the Gaussian distribution's parameters: μ_t and Σ_t.

With this formulation it is possible to tackle a variety of problems that require continuous random variables, but it has one intrinsic problem: not every real-world scenario can be assumed to have noise that is correctly modelled by a Gaussian distribution. This applies to both the external factors acting on the real state (the ones which are not part of the transition model's calculations) and the noise of the measurements. Regardless of these limitations, Kalman filters do perform well, and are widely used in practice. However, a more general model exists, that enables a variety of more complex noise models to be taken into account when performing filtering: *dynamic* Bayesian networks.

2.2.3 Dynamic Bayesian Networks

A dynamic Bayesian network, or DBN, is an extension of the Bayesian network that takes time into account. According to the previously described dynamic system, each step in time can be seen as a different Bayesian network, which represents how the model has changed after the new evidence has become available. This new model is a more general description of the already presented HMM and Kalman filters, which are just more restrictive cases of DBNs. For example, an HMM is a DBN having only one discrete variable.

The previous models favor simplicity and performance over the possibility of tackling a wider set of possible problems [7]. DBN, on the other hand, has an high computational cost exponential in the number of state variables [7]. To solve this problem, we rely on *approximate* methods, such as particle filtering.

Particle Filtering

Particle filtering is a family of algorithms whose working principle is based on the usage of samples to approximate the current state distribution. Each sample is a possible value that the random variable can assume.

The algorithms work as follows:

1. An initial set of N values is sampled from the prior distribution $P(X_0)$;
2. The next state x_{t+1} is computed for each sample x_t, using the transition model;
3. Each sample is weighted according to the new measurements and the sensor model;
4. Via resampling, a new set of samples is produced;

5. Steps 2 to 4 are repeated until an ending condition is met, usually the end of available data.

This simple algorithm family is very successful from a pragmatic point of view. However, its success is tied to the selection of its parameters which is far from simple and inexpensive. Starting from the initial samples distribution, passing through the usual transition and sensor models, and finally the weighting and resampling criterias, the great amount of parameters needs a deep knowledge of the domain and a great number of tries for correctly tuning the system.

As a final remark, a great advantage of particle filtering is that it effectively deals with multiple hypothesis even if they are greatly different from each other, since it can represent any probability distribution.

2.3 Bluetooth 5.1 localization

Bluetooth has been among the most important technologies for indoor localization ever since the introduction of beacons, establishing a de-facto standard for proximity solutions. There are many reasons for its success, the most important ones being its relatively low cost, and its presence on a wide variety of "smart devices", which could exploit it directly, without the need of additional hardware.

Recently, the Bluetooth SIG introduced a new standard: Bluetooth 5.1, which contains the specifications for an angle measurement system. This new technology only acts as an enabling mechanism which deals with the low-level parts of the stack, defining the protocols that allow a set of devices with defined roles to be able to compute the angle between them. The upper parts of the stack, where the angles are used for triangulation, is left to the developers.

2.3.1 Angle of Arrival and Angle of Departure

The Bluetooth 5.1 localization system can employ one of two possible methods: angle of arrival or angle of departure. The first one requires the receiving device to have a *multi-antenna array*, and enables it to compute the angle between itself and any transmitting device, as shown in Figure 2.2. The second one requires the sending device to have the multi-antenna array, computing the angle between itself and any receiving device, as shown in Figure 2.3.

Since the major inconveniences of implementing a localization system based on 5.1 revolve around the multi-antenna array, the two operating modes are crucial, as either one of them can perform better based on the scenario and its requirements. For example, for a robot localizing itself in a vast area, the angle of departure would be the best fit, as the robot will use the only antenna array, while the anchor devices would just work with regular Bluetooth hardware. On the other

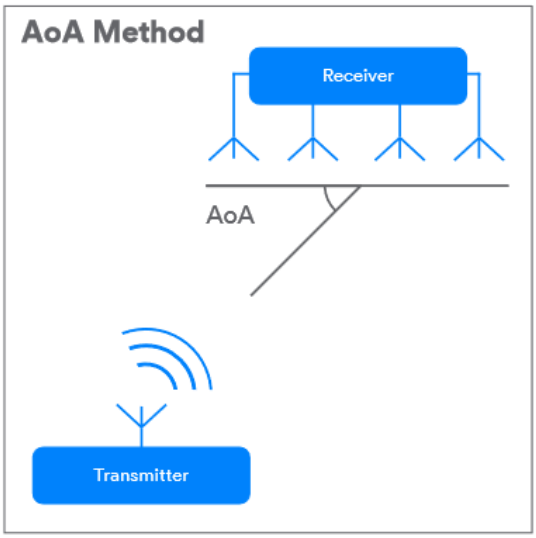

Figure 2.2: Angle of arrival method, with a multi-antenna array on the receiver. Source: [10]

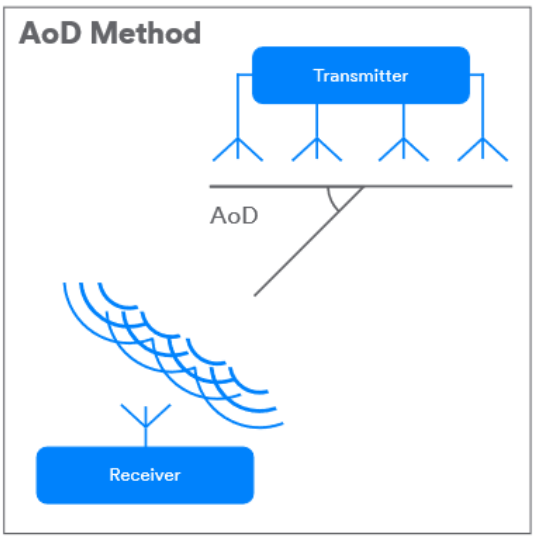

Figure 2.3: Angle of departure method, with a multi-antenna array on the transmitter. Source: [10]

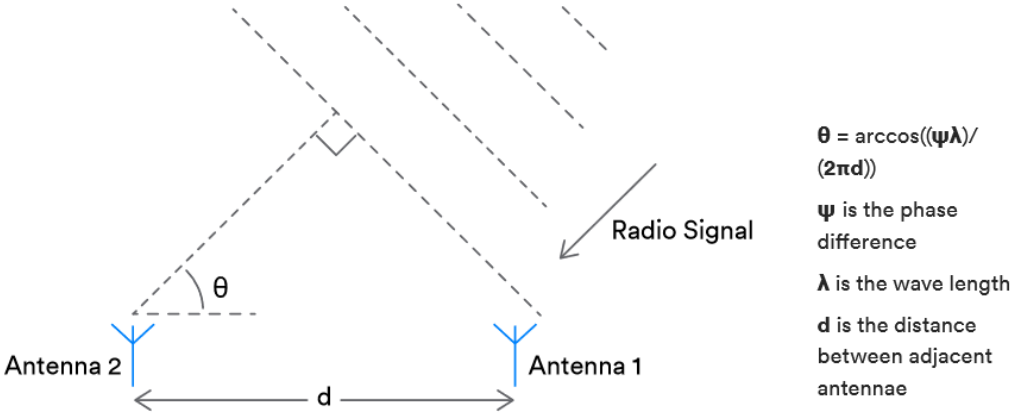

Figure 2.4: How trigonometry is used to compute angle of arrival. Source: [10]

hand, a system with a multitude of OBSERVED ITEMS and a relatively small area, where a few LOCATORS are sufficient, would benefit more from a angle of arrival mode.

2.3.2 Technical details

Radio direction finding has quite a long history, starting from Henrich Hertz's experiments which involved comparing the signal strenghts measured when the antenna pointed at different location. Since then, more accurate techniques have been developed, yet they just rely on fundamental wave properties and simple trigonometry. Bluetooth 5.1's angle detection is also based on such ideas, as it uses phase difference to determine signal direction. For brevity, only the technique based on the angle of arrival is explained, as the angle of departure is based on the same concepts.

Given a pair of receiving antennas at a fixed and known distance d, a third antenna emits a radio wave characterized by some phase values ψ_1 and ψ_2 based on the antennas and on their position relative to the transmitter and the wave length λ. Knowing the phase difference Ψ, antennas distance d and wave length λ, it is possible to use trigonometry to compute the angle θ

$$\theta = \arccos \frac{\Psi \lambda}{2\pi d} \tag{2.5}$$

as shown in Figure 2.4.

Although the only hardware change that needs to be done is related to the multi-antenna array, it is important to point out that the rest of the devices will

still need an upgrade to the 5.1 firmware, as it enables specific packet types and roles. More specifically, a new packet containing a constant tone extension (CTE) is introduced. It comprises of a regular packet with a final sequence of symbols representing the binary 1. This sequence is used for performing the phase sampling used in the previous calculations.

2.3.3 Performance

To the best of our knowledge, there is no research testing the performance of the Bluetooth 5.1 with a full-stack implementation. However, a research [5] based on the same working principle but on specialized hardware claims that it can achieve sub-meter accuracy.

Chapter 3

Design

In this chapter, we design a system to facilitate the analysis and tuning of indoor localization systems, based on the requirements and the theoretical aspects introduced in Chapter 2. Here, we clarify the meaning of names and verbs present in the requirements text, and define a logical architecture using the formalized vocabulary. The resulting architecture is described by means of UML diagrams, which are to be intended in a broader way as logical artifacts independent from the specific paradigm, and not implementation-specific diagrams deeply connected with object-oriented programming.

This project can be seen as composed of two major parts: the framework itself, and the Bluetooth 5.1 case study. This chapter is mostly concerned about the framework, but does make some analysis of the specific use case when necessary.

The remainder of this chapter is organized as follows: in Section 3.1 a formalized requirements description is given, in Section 3.2 the problem is analyzed, with particular attention to the matter of data fusion, and finally, in Section 3.3, an architecture of the analyzed system is presented.

3.1 Requirements Analysis

A requirements text is introduced to clarify which are the specifications of the system, then it is analyzed and formalized to reduce the possible natural language inconsistencies and misunderstandings. The text is as follows:

> The "AoA localization system" is required to be a framework that enables the execution of a data fusion algorithm which uses angle of arrivals from a range of locator devices to compute the position(s) of one or more target items. The data can either come from real devices connected to the system, old data that was recorded and has been played back, or simulated data.

> The system will be used for fine-tuning the parameters so that the specific localization service's performance can be improved.

This minimal requirements text is the base for the discussion presented here, in which every aspect of the described system is analyzed.

The requirements show that this is an heterogeneous distributed system, which requires us to identify its participants. First, we have the ObservedItem, which is an item whose location is unknown. The objective of the system is to find ObservedItem's location. Then, the Locator is the entity responsible for measuring an angle between itself and one or more ObservedItems. Finally, the LocalizationService is the one accumulating all the necessary information and performing the data fusion step. These three entities are the very core of the system, as both the simulated and real-world cases need to deal with those entities.

The ObservedItems and the Locators are both characterized by their location, which makes them *situated*. On the other hand, the LocalizationService is not required to have any specific position in space, and as such it is a disembodied computational node.

These few specifications clarify the meaning of the requirements text, providing a formalized vocabulary to refer to the main entities involved. Moreover, they suffice to identify a set of challenges that need to be addressed. In the following sections, these problems are analyzed and possible solutions are proposed.

3.2 Problem Analysis

The requirements analysis identifies two main threads that need to be further studied and discussed: data fusion applicability and the distributed nature of the system.

3.2.1 AoA and data fusion

The requirements identify the specific problem of localization through angle-of-arrival usage, which is characterized by other aspects based on the specific domain. For example, a scenario in which a car is the ObservedItem the system could have additional information about the speed at which the car is moving. Here, we discuss the problem according to its base specification, then briefly discuss the specific Bluetooth use case.

Before diving deep into data fusion, it is important to understand which information are actually flowing into the system, and what are their characteristics. The "location" is a central information to the system, and as such we need to define how it should be represented. A standard way of doing so, is using cartesian

coordinates in an arbitrary reference system. Another piece of data that is crucial is the measurement of an angle. Using a tuple of numbers to represent it should accommodate both degrees and radians in different dimensions.

Given the problem of localization via triangulation, to find the position of an ObservedItem we need at least two Locators, their positions, and the rays passing through them and the ObservedItem. Such rays are identified by one point in space and, depending on whether we are working in two or three dimensions, either one or two angles. This is the minimum required information for performing data fusion on the problem at hand, but a better result can be obtained if a greater number of entities are involved. As previously said, in ideal conditions all the rays would just meet in one point: the ObservedItem's location. However, considering the measurements are noisy, it is more likely that there are multiple intersection points, and even pairs of rays that do not intersect at all, which implies data fusion is *needed* to have any useful results. To identify the best suited model, it is crucial to know what kind of noise is interfering with the sensors, and what external forces act on the ObservedItem to change its position over time. However, these are case-specific information, and thus should not influence any decision about the structure of the framework, which should be general enough to accomodate the needs of all the possible scenarios. Summarizing, the data fusion algorithm to be employed is deeply connected to the specific case, even though some solutions can be partially employed for different scenarios.

The Bluetooth case study can indeed be analyzed in a more specific manner, soving both its problem, and providing a test case used to validate the correctness of the framework. First, the noise affecting the sensors is not known a-priori, so a solution like Kalman filters would not be suitable, as it assumes the noise is Gaussian. Then, the ObservedItem does not have any on-board sensors capable of tracking how it moves in space (e.g. no odometer), or at least such information is not integrated into the system. One assumption that can be made is that it behaves as an inertial body, making use of the estimated movements in space to predict its speed. This can be said because of the scenarios considered for the usage of the system, which mainly comprises of wearable devices and smart vehicles.

All these considerations about the uncertainty of this case study indicate that particle filtering should be used. The first advantage is the possibility of keeping multiple hypothesis at the same time, considering a non-gaussian random variable distribution. This is very important because the noise affecting the angle measurements could make the different rays meet in points which create more than one cluster, and this technique would accomodate such an event by keeping the different particles around the various clusters at the same time. Moreover, such a complex distribution would be difficult and error-prone to update explicitly using a random variable, while it can be handled in a relatively simpler way by updating

the particles.

Additional information about the actual implementation of the particle filtering algorithm for the Bluetooth case study is given in Section 4.2.

3.2.2 Heterogeneous distributed system

Heterogeneous distributed systems pose a wide range of problems, the most important ones being coordination and interfaces definition. The former is deeply tied to the distributed aspect, the latter with the heterogeneous one.

Coordinating a distributed system requires some protocol that the different nodes of the network need to follow. The protocol enforces message ordering, makes sure that all the relevant information is being exchanged, and defines the overall flow of the system.

Node heterogeneity requires the different nodes to be able to communicate with each other. The commnication should abstract from the underlying physical nature of the device, using a common technology for exchanging information between all nodes of the system.

3.3 Architecture

After analyzing the requirements and the problems, the nature of the system is clear, and its critical aspects are identified. With these information, an high-level architecture can follow as a logical consequence.

The first aspect that directly derives from the problem analysis is the need of using a paradigm that can handle a distributed heterogeneous system in the correct way. The Actor paradigm[1] is the model of choice, as it is message-based and directly addresses the previously discussed problems. This is optimal for the heterogeneous nature of the system, as it provides a standard messaging interface that is independent from the location and nature of the devices. Another important aspect that it deals correctly with, is the protocol definition, thanks to both the fact that it is message-based, and the run-to-completion semantic. Moreover, the Actor paradigm is particularly suited for describing a system from a high-level perspective, making use of UML diagrams.

This section does not depend on any particular programming language, it just provides an architecture that can be implemented with the best suited technology.

The remainder of this section describes the system through its three main dimensions: structure, interaction, and behavior. The order of discussion is important as we first need to define which actors are part of the system, then explain the way they interact with each other, and finally describe how each one singularly behaves to accomplish the whole system's goal.

3.3.1 Structure

When describing an Actor-based system, the structure is greatly influenced by the main actors and their relationships. The system is also heterogeneous, so the coarsest grained structural description should take that in consideration by defining two main parts: the situated system, and the disembodied one.

The situated part comprises of ObservedItems and Locators. These can be present in a variable number of instances, with a required minimum of one ObservedItem and two Locators. Each situated actor is in charge of handling the communication with the other, creating an abstraction over the embedded nature of the device and its internal state. The advantage of using the actor paradigm is that we are not concerned with knowing the physical topology of the system, as two different Locators could run on the same board that manages two different antenna arrays, and the system would still work the same way as it would if the Locators were on two different physical devices.

The disembodied part of the system has the LocalizationService as its main actor, but since it is the part which computes the final result it is likely that it will be also concerned with communicating this information to an interested user. That can be done in several possible ways, but is a lower level, technology-dependent detail discussed in Chapter 4.

Once the main entities are specified, it is possible to discuss the main interfaces that define the relationships between them, as shown in Figure 3.1. First, the situated system is characterized by the possibility of exchanging some low level data, as it happens for example in the Bluetooth use case, where the ObservedItems send the CTE packet to the Locators, which can compute the angle with that information. Then, it is required that the LocalizationService provides a mean to register the angle which can be use by Locators to communicate the perceived data.

3.3.2 Interaction

An Actor-system is message-based, which means its fundamental way of communicating is via message exchange. All of the discussions about interaction are based on message exchange patterns to deal with unknown delay, with the end goal of synchronizing the system. The triangulation problem only requires the exchange of a few messages, at least from an high-level perspective. We can define two main steps: the initialization and the working loop.

When initializing a distributed system like this, the main concern is synchronization, which means we want all of the Locators to be ready to work before we start the actual localization. This requires the LocalizationService to know which Locators should be part of the system, then wait for a message by them

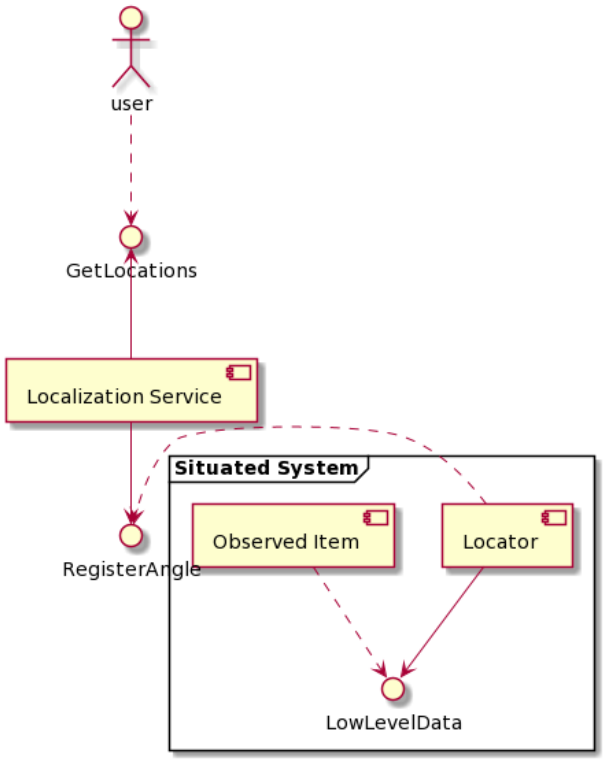

Figure 3.1: A component diagram showing the overall system structure and its main actors.

that communicates they are ready. Once all of the LOCATORs are ready, the LOCALIZATIONSERVICE can broadcast a message to the LOCATORs, indicating that the system is ready to work.

The actual localization is a rather simple loop, where each OBSERVEDITEM sends the low level data if that is required, and upon receiving it each LOCATOR computes the angle and communicates it to the LOCALIZATIONSERVICE. At that point, the system can start a new cycle of the loop, after the LOCALIZATIONSERVICE optionally communicates the currently estimated location to some output node.

Even though the messages exchange is rather simple, two considerations need to be made. First, the OBSERVEDITEMs are not part of the first initialization step, as their presence in the system can be dynamic. This means that an OBSERVEDITEM can enter and exit from the situated system area, possibly because it is temporarily turned off, and that any number of OBSERVEDITEM can be plugged into the system at runtime. The only requirement is that each OBSERVEDITEM has a *unique* ID, which must be included in the low level data message, or that it has some other way of being uniquely identified (for example, that could be useful in a use-case that makes use of computer vision to compute the angles).

The other important remark is that there is the need for some synchronization during the loop: if one LOCATOR misses some messages and then communicates a new angle of arrival to the LOCALIZATIONSERVICE it must not be matched with the previous angles computed from the other LOCATORs, because these would be angles that refer to different moments of time. How this problem is solved can be a low-level detail, but it is relevant to discuss it here as one of the possible solutions could also be adding some metadata to the angle of arrival message (e.g. a timestamp).

A visual representation of the interaction explained here is available in Figure 3.2.

3.3.3 Behavior

The behavior of each main actor is described by means of finite state machines, as depicted in Figure 3.3, and in accordance with the information about the interaction described in the previous subsection.

The OBSERVEDITEM simply has one logical state in which it is active and, if needed, sends data to the LOCATORs. The decision of keeping this actor rather simple is a conscious effort in handling the complexity on the framework side, limiting the embedded devices' effort to the low-level parts.

The LOCATOR starts in an idle state, and after the registration to the service is successful, it loops through two states: waiting and sending. It alternates between the two each time new data arrives from the OBSERVEDITEM and each time it is

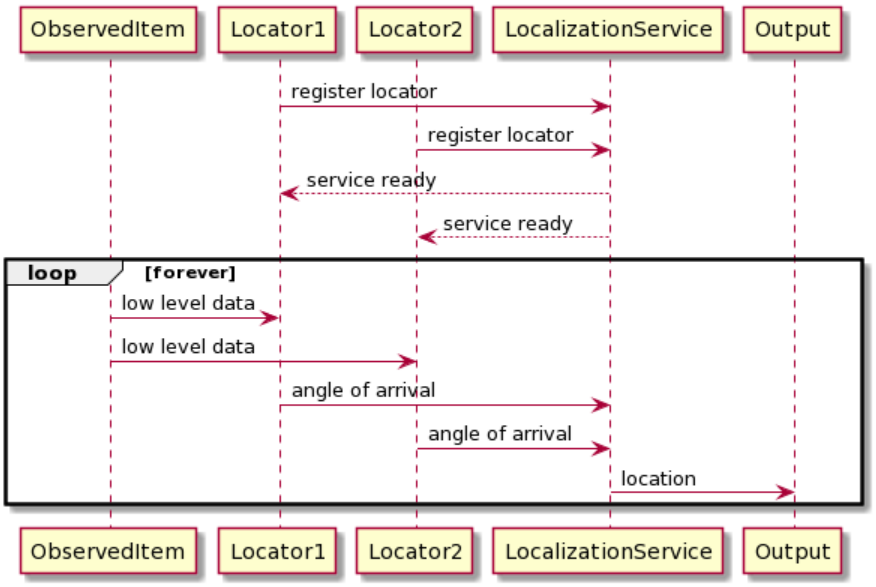

Figure 3.2: The sequence diagram describing the overall interaction flow.

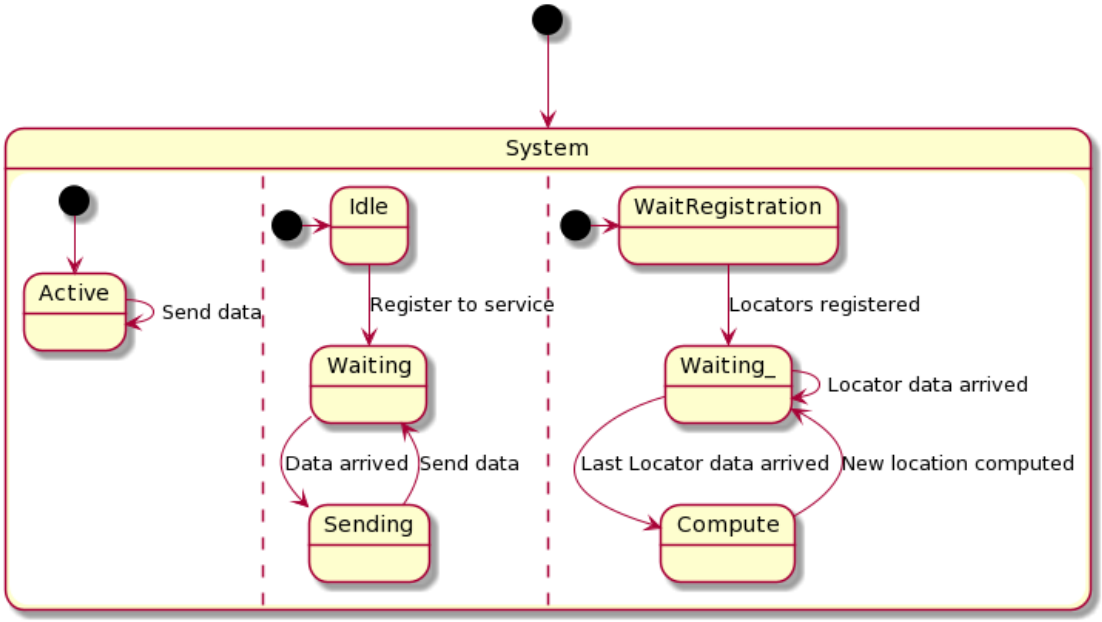

Figure 3.3: The state diagram, divided in three lanes. The leftmost lane contains the ObservedItem behavior, the central one describes the Locator, and the rightmost is for the LocalizationService.

sent to the LocalizationService. Any policy about waiting for more data or not sending the data can be enforced in the state that best fits the case.

Finally, the LocalizationService acts similarly to the Locator, with an initial state that waits for the registrations for the service, and a loop. The biggest difference here, is that the way steps are handled is directly addressed: the LocalizationService stays in the waiting state until all of the Locators send the angle for the current step. Once that happens, the LocalizationService transitions in the compute state, where it performs the data fusion task, and at the end goes back to waiting for the next step's angle messages to arrive.

Chapter 4

Implementation

This chapter is concerned with explaining *how* the system has been implemented and the technologies in use. A reader that desires to use or extend the framework, can find useful insights here. The source code of the project is available at its GitLab repository[1].

Based on the discussion of the problems in the previous chapter, the programming language of choice is python. The main reason for this is that the end goal of the system is to perform fine-tuning of data fusion algorithms in specific scenarios, which can be a strategic part of the development of a system, so the possibility of shortening the time-to-market, or to rapidly evaluating a prototype are the top priorities. Python is particularly goof in this case thanks to both its extremely succinct syntax and the abundance of libraries, especially in the field of mathematics, which not only speed up the first stages of development, but also ensure that they are error-free, as the most complex aspects of the data fusion part might be already handled by some external library. On top of that, python is very popular among professionals whose primary job is not programming (e.g. mathematicians), and has a good predisposition to both being a scripting language, and performing data analysis and visualization. That being said, python is not the only suitable language for a system of that type, and any other programming language would have been appropriate for developing the system at hand.

In this chapter, data fusion is mostly discussed referring to particle filtering in particular, because of the use case using it. The relevant encapsulation mechanisms that make the framework robust and easy to use and expand, are described later in Section 4.2.

The remainder of this chapter is organized as follows: first, in Section 4.1, the actor system's low-level details are explained, then in Section 4.2, the mathemati-

[1] `https://gitlab.com/pika-lab/theses/thesis-nemati-ay2021/particle-filter-localization`

cal model is interpreted for the Bluetooth use case, defining the specific algorithm. Section 4.3 presents an overview of how the system handles simulations, and finally in Section 4.4 the physical part of the use case and its integration with the framework are described.

4.1 Actor System

The actor system implementation is based on the pykka library[2], version `2.0.3`. It is a python library that provides a simple implementation of the actor model, proposing two main working interfaces: basic actors, and proxy actors. The former explicitly uses the actor model's primitives, like send and receive, the latter provides an abstraction that incapsulates the behavior of the actor in an object that has a separate method for each possible action. In the case of a proxy actor, the different methods return a future.

While the proxy can be a fairly useful abstraction to keep the code clean, the basic actor implementation is a more explicit representation of the distributed system, so it is the standard way actors are implemented in this project. However, any extension to the system might use the proxies, as they are just wrappers and can thus be used in conjunction with the basic actors.

The use of pykka provides important support for dealing with the challenges of concurrent systems, but on the other hand it is not a very mature technology, and as such it lacks some fundamental features. The most important one is the absence of the "become" primitive, necessary for a good mapping between the state diagrams and the implementation. While it could be possible to make a brand new actor every time the state changes, it would be cumbersome and error-prone to work that way. In the project, a BaseActor class is provided to compensate for the lack of the "become" primitive. This class handles the state change by keeping a state variable and a dictionary that maps each state to a function handling a message. That way, each state is modelled by a function that specifies how the actor implements the "receive" primitive, and when a message is received, the function that needs to be called is selected depending on the current actor's state.

Using that mechanism, each of the actors described in the architecture is implemented, using a more fine-grained state diagram as a reference. In the following paragraphs, each actor is described.

Before looking at each single actor, it is necessary to first describe how the actual coordination protocol is defined. Since the main focus of the framework is on simulation and fine-tuning, the actual mechanism for synchronizing each actor has been chosen prioritizing simplicity over partition tolerance: a centralized

[2] `https://www.pykka.org`

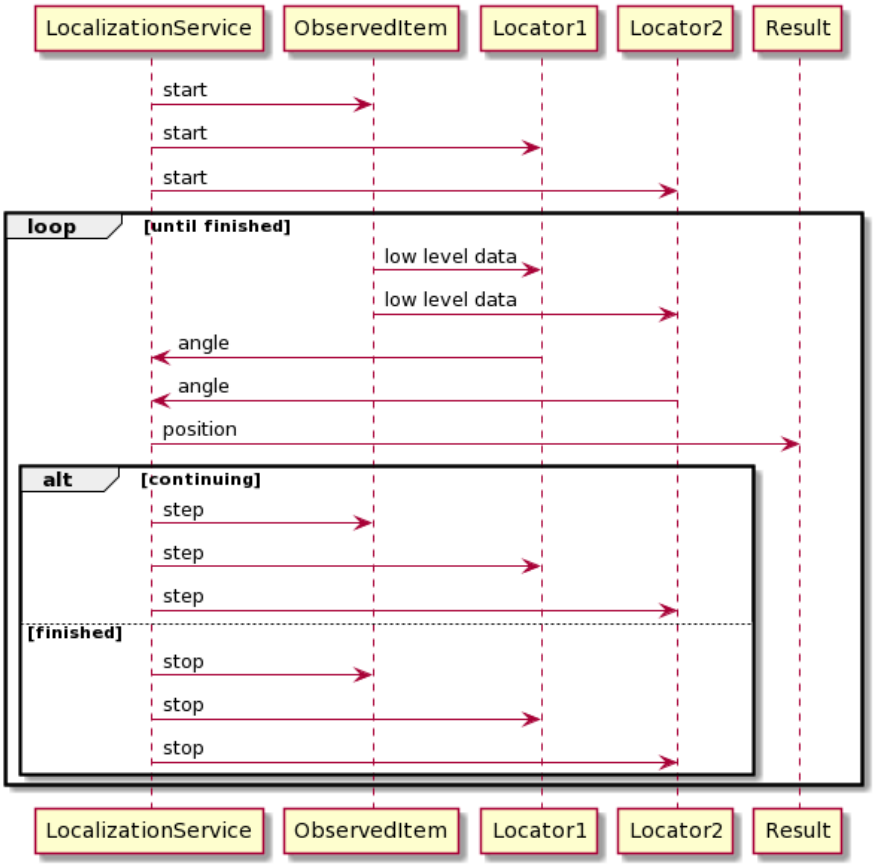

Figure 4.1: The sequence diagram describing the overall message exchange revolving around the "step" and "stop" messages for coordination.

solution based on explicit messages stating the transition to the next step of the computation. More specifically, the LocalizationService first notifies all the other actors about the start, then enters a loop in which it first waits until it receives all of the messages from the other actors, then computes, and finally sends out a "step" message when it is ready to transition to the following time slice. When the computation must be interrupted, the LocalizationService sends a "stop" message. This overall flow is shown in Figure 4.1.

Observed Item actor. This actor's behavior is a bit more detailed if compared to what is described in the design phase, as it needs to both handle the step and stop commands, and be easily used in simulation scenarios. It has three possible states: idle, waiting, and send low level data. The idle state is useful for handling the life-cycle of the ObservedItem, as it is possible to react to "start" and

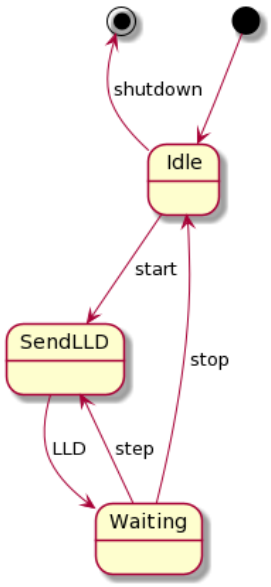

Figure 4.2: The state diagram of the ObservedItem. "LLD" is short for low level data.

"shutdown" messages, and it is the state in which the actor transitions when the "stop" message is received. The send state is present on a logical level, but is not necessary to include it in the implementation, as the run-to-completion semantic of the actor system already handles the required behavior. In case no low level data needs to be sent to the Locators, the send state can simply be ignored. Finally, the waiting state is crucial for the synchronization with other actors.

A visual representation of the ObservedItem's behavior is depicted in Figure 4.2.

Locator actor. The Locator actor's behavior, in terms of states and transitions, is very similar to the previously described ObservedItem. This is because they are treated in a uniform way from the LocalizationService, which uses the same kind of messages to synchronize both types of actors. What changes is the actual behaviour in the working state, which, in that case, is the read angle state. Here, it is necessary that this state is present in the implementation as well, as it is used to wait for the low level data coming from the ObservedItems. In case no low level data is needed, a simple self-message can be used. The state diagram is shown in Figure 4.3.

Localization Service actor. Its LocalizationService serves two important functions: it both performs the computation and synchronizes the whole system.

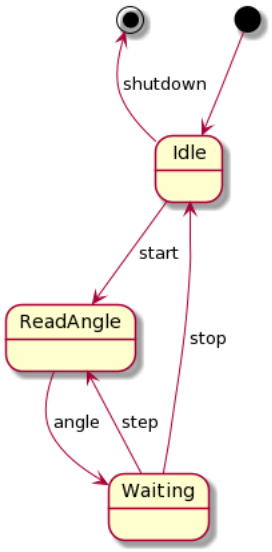

Figure 4.3: The state diagram of the LOCATOR.

The state diagram is presented in Figure 4.4. The computing state does not necessarily need to be explicitly present in the implementation, however, for complex computations that might require usage of external services it can be fairly useful.

4.2 Particle Filtering

The use case's analysis in the previous chapter identifies particle filtering as the appropriate data fusion algorithm to deal with the specific problem's characteristics. Being it a *family* of algorithms, it is necessary to define how it actually works, in particular describing the chosen transition model, sensor model, and re-sampling strategy.

Transition model. Usually, the transition model takes into account some information about how the state is changing. An example in the domain of locating a moving object is a robot moving in space which has an odometer on board, or that has a battery level indicator. The former directly provides the necessary information to update the particles state, the latter can influence the transition model if the assumption that it goes slower as the battery depletes can be made. However, the OBSERVEDITEMS do not provide any information about their current state themselves. One assumption that is possible to make is that they move in an *inertial* way, so knowing the velocity at the previous time is enough for computing the transition model.

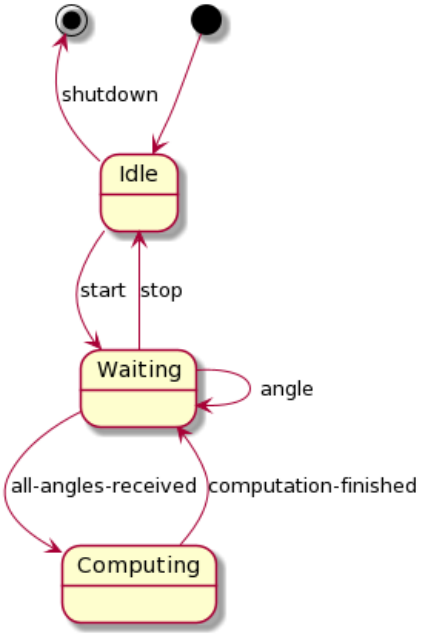

Figure 4.4: The state diagram of the LocalizationService.

An important remark is that the transition model is based off of a probability density function, which means the particles are not updated using the exact equations of motion. Instead, a probability density function that has its highest probability values in a vector that matches the exact equations of motion is used for updating the particles' state.

To infer the approximate velocity of the ObservedItem, the center of mass of the particles is computed at the previous step and the current step. The difference between the two provides a reasonable vector to be used for updating the transition model.

The transition model is *dynamic*, as it changes at each step of the computation based on the inferred velocity. The probability density model used is a Gaussian, leveraging the fact that the velocity should be uni-modal.

Sensor model. The sensor model is used to assign a greater weight to the particles that better fit the angles recorded by the Locators. An implementation that is scalable with respect to the number of Locators is based on the average minimum distances between a particle and each ray corresponding to a Locator and its angle read. The random factor is already taken into account by the natural noise affecting the angle measurements.

Re-sampling strategy. Re-sampling is based on the weights assigned by the sensor model, which get normalized and used as weights for randomly choosing N elements from the current particles, which has N elements itself. This means that the total number of particles remains the same, which is computationally intense, but grants a robust mechanism to deal with very noisy spikes. Moreover, particle filtering works better with a great number of particle, and the overall complexity stays stationary over the course of the computation.

Without any additional information, the best suited initial distribution is generated by uniformly placing samples in a given bounding box. This is just the general case, however, some ad-hoc initial distributions can improve the results in both terms of accuracy and number of iterations needed to form the first clusters.

4.3 Simulation

The framework has a major focus on simulation, which, however, is not discussed in detail in the previous part of the thesis. This is because the system is modelled in such a way that accommodates both real and simulated data using the same abstractions and overall structure. More details on the simulation capabilities of the system are presented in this section, discussing a way of generating data at run-time, and a way of generating the data before the computation (possibly using some external tool) and then replaying it inside the framework. Both of these modes of operation are based on the usage of custom actors which inherit from the original ObservedItem.

The simulated observed item's actor takes an ordered list of positions as input, and for each Locator computes the data that needs to be sent so that the resulting angle is affected by some noise, but still based on the actual one. The implementation provided in the framework applies a Gaussian noise to the simulated data, but it is possible to plug any other kind of distribution to be used for sampling the value. This way of simulating is particularly suited for a scenario in which the focus is on the correct tuning in specific movement patterns.

The replay observed item's actor works in a similar fashion, taking as an input the raw data, and playing it back as the simulation proceeds, without altering it in any way. This approach is useful when dealing with real data for fine-tuning, or when it is more convenient to generate the simulated data in another tool, and this framework is used for the fine-tuning step.

Using the already provided functionalities, it is rather simple to setup a simulation environment, with an arbitrary number of Locators and ObservedItems. This is useful for performing various experiments and not only fine-tuning the data fusion algorithm's parameters, but also the number of devices used.

Moreover, the reproducibility of the simulation is granted by the option to specify a custom seed that is used throughout the whole framework for generating every random number. The deterministic behavior of the simulation is an important feature as it makes it possible to compare various parameters' values in an accurate way, since they would work on exactly the same numbers.

4.4 Situated System

The process of choosing a development kit that suits the Bluetooth use case's requirements could be a chapter of its own, but for the sake of only discussing computer-science related topics, here we only described the devices which have been used. The main reason for the choice is the compatibility with the BT 5.1 standard, and the availability of a custom antenna array ready to use for the use case.

The devices of choice consist of 3 LAUNCHXL-CC26X2R1[3] boards, and 2 BOOSTXL-AOA[4] antenna arrays, all by the producer Texas Instruments[5] (TI). With the given hardware, two Locators and ObservedItem have been implemented, relying on the dedicated libraries for the low-level embedded system part.

The software that runs on the boards must be written in the C programming language, making use of some proprietary libraries. Texas Instruments provides ready-to-use software that can be run on the physical devices, which have been used. The given software is set to use UART ports to make the embedded device communicate with a central PC node, as shown in Figure 4.5.

The "Node manager" provided by TI is written in python, and is fairly easy to edit. For the specific use case, it has been modified to record the angles so that these could be later replayed in the framework.

The details about the experimental setup and yielded results are presented in Chapter 5.

[3] https://www.ti.com/tool/LAUNCHXL-CC26X2R1

[4] https://www.ti.com/tool/BOOSTXL-AOA

[5] https://www.ti.com/

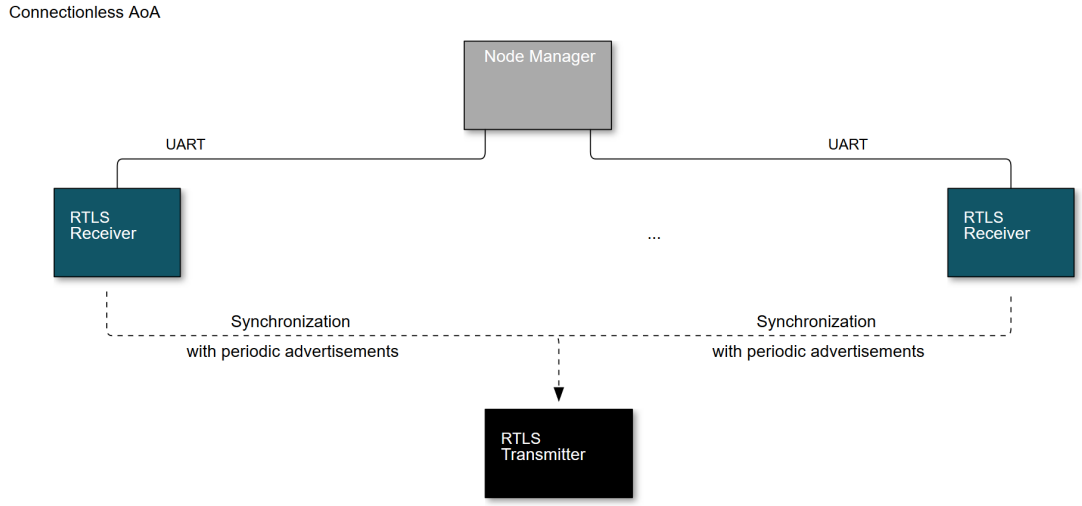

Figure 4.5: An architectural overview of the situated system. The "transmitter" is the ObservedItem, the "receivers" are the Locators.

Chapter 5

Validation

A framework's value is based on its impact on the development of a specific application, while a localization system's value lies within its performance. The former can be expressed in various metrics such as development time, and a first evaluation can be made with respect to the Bluetooth use case. However, more applications need to be developed to really test the strength of the framework. The latter is application-specific, but the same metrics apply to all the systems.

The remainder of this chapter is organized as follows: in Section 5.1 the tested scenarios are explained and evaluated, in Section 5.2 a brief overview of the code quality tools used is given.

5.1 Experimental Results

When evaluating a localization system, there are many performance indicators that need to be taken into account[3]. They are: accuracy, precision, latency, energy consumption, cost, complexity, coverage, robustness. While all of these are very important, in this section we are only concerned with the first two indicators, as they are able to tell if the parameters fine-tuning is yielding better results or not. The remainder of the indicators can be evaluated independently of the framework, even though it can have some influence on them. For example, tuning the system to achieve some target accuracy and precision with the least number of LOCATORs can have an impact on the overall cost and energy consumption.

Accuracy is an indicator of the error distance between the estimated and actual positions. Using data fusion, however, the location of an OBSERVEDITEM is not just one point in space, but a probability density function which assigns higher probabilities to certain points, and lower values to others. This can be solved in different ways, but since the actual localization process requires *one* position, and should not be concerned with the mathematical technicalities, here accuracy is

Listing 5.1: Setup of parameters in the framework for fine tuning

```
position_1 = (0, 21)
position_2 = (60, 21)
position_3 = (30, 21)

positions = [position_1, position_2, position_3]
particle_numbers = [10, 50, 100, 500]
sigmas = [0.0, 0.4, 0.8, 1.2, 1.6, 2.0, 2.4]

settings = generate_settings(positions, particle_numbers, sigmas)

for setting in settings:
    particle_filtering_simulation(setting)
```

computed with respect to the global maximum of the probability function, or the highest-weighted particle in case of particle filtering.

Precision can be expressed as the percentage of correct localizations, where "correct" is relative to a desired accuracy value. The target accuracy should be based on the domain and requirements constraints.

Examples of experiments evaluation for both a simulated case and the Bluetooth use-case are presented here. The two experiments are connected as the simulation is an initial step in exploring the fine-tuning of a particle-based system, and the use-case is a test scenario for the optimized parameters found during the simulation.

Throughout this section, space measurements are to be intended in centimeters, and the studied value is the mean distance between the points proposed by the system and the real positions of the ObservedItem.

5.1.1 Simulation

There are a variety of parameters that can be modified to improve the accuracy of the system, and testing all of them can be quite challenging. However, the simulation framework provides an easily customized environment which allows a rather complete experimental setup to be employed. The experiments shown here have been set up with a combination of parameters as easily as shown in Listing 5.1.

The major factors impacting the end result are the number of Locators, the number of particles, and the probability density distributions used for the transition model. Different possible values are chosen for each of these parameters, then each combination is played in turn, and the results are compared. The scenarios have an impact as well, so the previous combinations are tested in two different

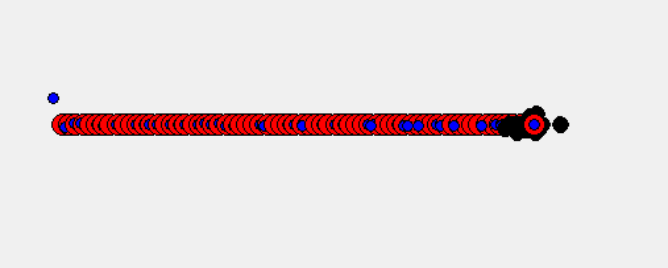

Figure 5.1: Visual representation of the particle filtering applied to simulated data where the ObservedItem moves at constant speed.

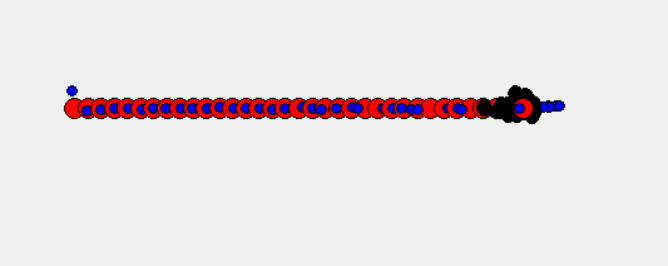

Figure 5.2: Visual representation of the particle filtering applied to simulated data where the ObservedItem moves at constant speed and then stops.

cases: with an ObservedItem moving with constant speed, and with it first moving at constant speed, and then stopping. A visual representation of the simulations is provided in form of screen captures in Figure 5.1 for the scenario of an ObservedItem moving at constant speed, and in Figure 5.2 for the scenario in which it first moves at constant speed and then stops. In these figures the points in black represent the current particles, the red ones are the real positions, and the blue ones are the best hypothesis, one for each time frame.

The different values used for tuning are: 2, 3, 4 and 5 different Locators; 10, 50, 100, and 500 particles, and 0.0, 0.4, 0.8, 1.2, 1.6, 2.0, and 2.4 for the standard deviation of the transition model's Gaussian. The results are presented in Figure 5.3 for the case in which the ObservedItem moves at constant speed and in Figure 5.4 for the one where it first moves at constant speed and then stops. Each figure comprises of 4 different graphs; in order from left to right, and top to bottom, they are for 2, 3, 4, and 5 different Locators. Each line in each graph represents the mean distance between the real position and the one guessed by the system based on how many particles are employed: the solid line represents 10 particles, the dashed one 50, and the dash-dot one 100, and the dotted one 500. The graph shows how the mean accuracy varies based on the value of sigma for the different scenarios.

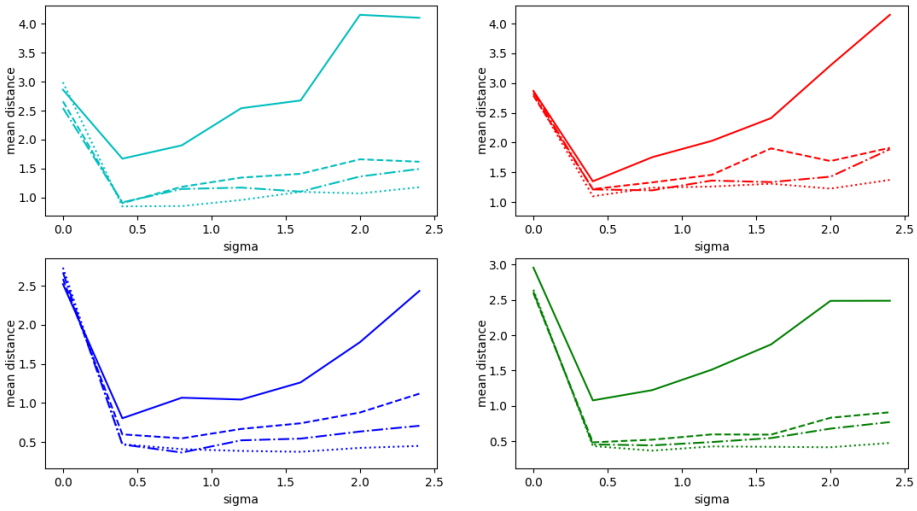

Figure 5.3: Average distances for different parameters values, with the ObservedItem moving at constant speed. The four graphs each represent scenarios with a different number of Locators: cyan has 2, red has 3, blue has 4, and green has 5. Each line in each graph represents the number of particles employed: the solid line represents 10 particles, the dashed one 50, and the dash-dot one 100, and the dotted one 500.

Looking at the graphs, it is possible to state that both the constant speed and the one that goes at constant speed only at first, follow the same overall pattern: increasing the number of particles and Locators improves the result, and small values of sigma grant a more accurate estimate. While increasing the particles count, the computational cost must be taken into account, as increasing the number even further would degrade performance of a real-time localization system. The case analyzed in Figure 5.4 has a less predictable behavior, most likely caused by the sudden stop of the ObservedItem. The most notable difference is, in fact, the higher overall mean distance, which indicates it is more difficult to predict the real location in that kind of scenario, but ensures the parameters make a similar impact on the result even in different scenarios.

Another interesting difference between the two cases is that the peak in average distance is for sigma = 0.0, while for the second this is not always the case. That can be caused by the ObservedItem stopping at a certain point, which is better modelled by a transition model that favours a stationary position of the particles.

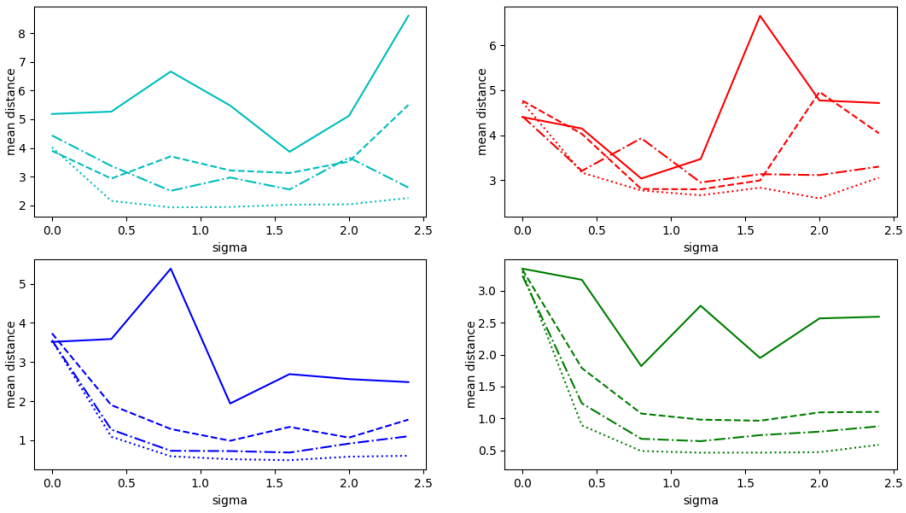

Figure 5.4: Average distances for different parameters values, with the ObservedItem moving at constant speed for the first half, and standing still for the second half. The four graphs each represent scenarios with a different number of Locators: cyan has 2, red has 3, blue has 4, and green has 5. Each line in each graph represents the number of particles employed: the solid line represents 10 particles, the dashed one 50, and the dash-dot one 100, and the dotted one 500.

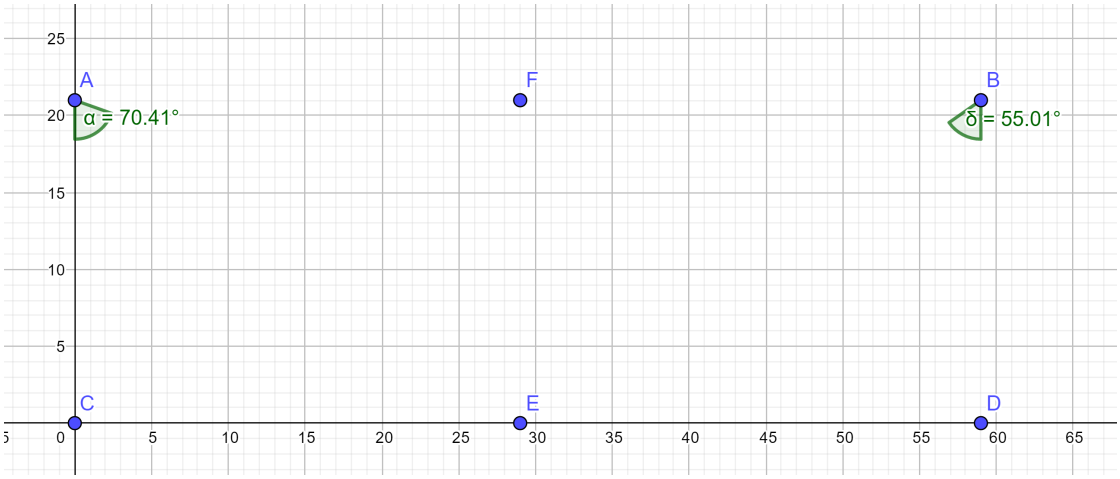

Figure 5.5: Position of the two LOCATORs (A and B), and the points through which the LOCATOR passes through in the different scenarios (C, D, E, and F).

5.1.2 Physical Experiments

The physical experiments based on the Bluetooth use case provide the angle measurements to be replayed in the framework. Because of physical and time constraints (mainly related to the Covid-19 pandemic), only two LOCATORs have been employed, and few, imprecise scenarios are taken in consideration. Three scenarios have been selected and analyzed, all of which are characterized by the two LOCATORs standing still in positions $(0, 21)$ and $(60, 21)$, respectively. First, the OBSERVEDITEM moves in a straight line from position $(0, 0)$ to $(0, 60)$, then it stands still in $(29, 0)$, and finally it moves along two lines: first $(0, 0)$ to $(30, 21)$, then to $(60, 0)$. It is important to remark that the precision of the OBSERVEDITEM's movements is limited, so it adds up to the expected noise that is already taken into account for the angle measurements.

The physical setup is depicted in Figure 5.5, where the LOCATORs correspond to points A and B, and the previously described coordinate the OBSERVEDITEM passes through are labelled C, D, E, and F. The results with different parameters are shown in Figure 5.6 for the case in which the OBSERVEDITEM moves at constant speed, and in Figure 5.7, where it stands still for the whole duration. A first visual feedback of the process is shown in Figure 5.8, where the OBSERVEDITEM stands still, and in Figure 5.9 where it moves along two lines. As the simulation proved, with a greater number of particles a better result is achieved. The overall mean distances are rather high compared to the simulation, but real data has more unstable noise interfering with its sensors and real movements. Overall, errors below one meter are achieved, which is an improvement over other state-of-the-art

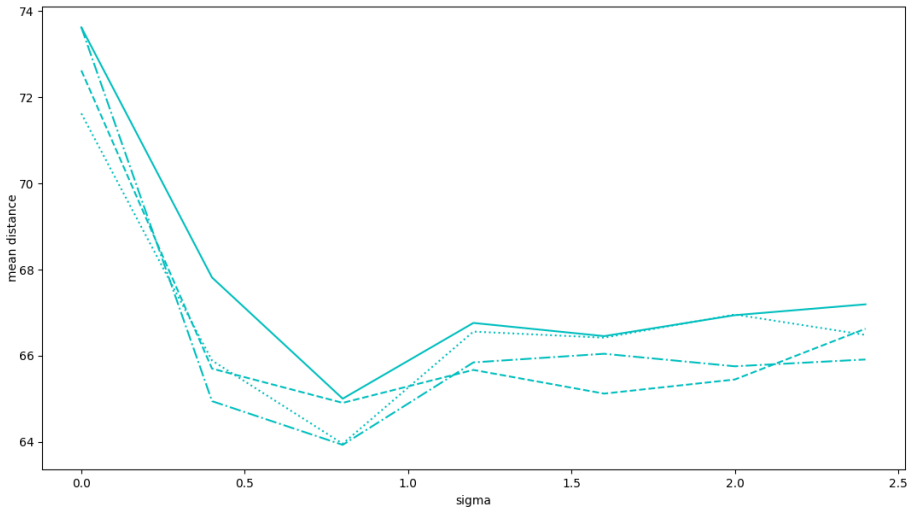

Figure 5.6: Average distances for different parameters values, using real data in which the ObservedItem moves at a constant speed. Each line in the graph represents the number of particles employed: the solid line represents 10 particles, the dashed one 50, and the dash-dot one 100, and the dotted one 500.

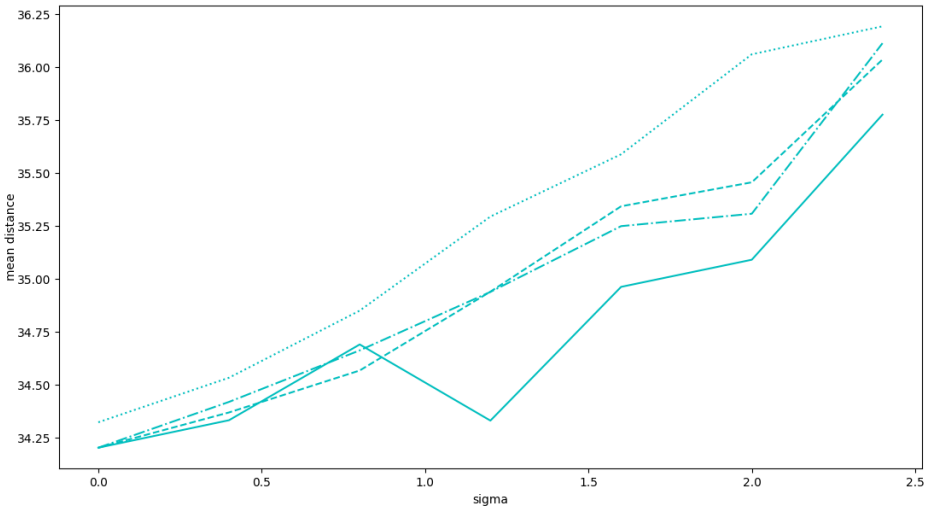

Figure 5.7: Average distances for different parameters values, using real data in which the ObservedItem stands still for the whole duration. Each line in the graph represents the number of particles employed: the solid line represents 10 particles, the dashed one 50, and the dash-dot one 100, and the dotted one 500.

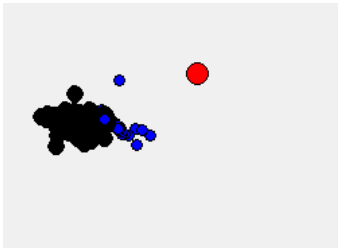

Figure 5.8: Visual representation of the particle filtering applied to real world data where the ObservedItem stands still.

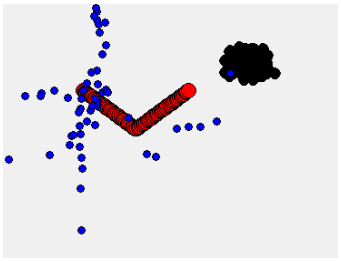

Figure 5.9: Visual representation of the particle filtering applied to real world data where the ObservedItem moves along two lines.

solutions.

An interesting fact is that the graph in Figure 5.7 shows the lowest mean distances for sigma = 0.0. Since the scenario considers an object standing still for the whole duration, this is a special case in which the transition model is likely to introduce additional noise instead of coping with it. This aspect has been anticipated by the experiments carried out in the simulation. The usefulness of the system lies also in these scenarios, where a particular case can be identified and analyzed to understand how different parameters affect the accuracy in specific settings.

When performing fine-tuning, small adjustments can be made, which have a relatively minor impact on overall accuracy, but such improvements are very consistent among different cases, and as such it is wise to exploit them. Another use of the system is to test completely different models, when it is not simple to choose the model that best approximates the system being studied. Testing a variety of different models against real data can provide an easy way of deciding which one should be applied, and with which parameters.

5.2 Code Quality

Code quality is important in every software product, but it is even more so in this case, both because of it being a framework (thus imposing the coding standards to the systems made with it) and because the python language is very flexible. When a framework has poor code quality, the users of the framework are going to produce similar results, so the methods for ensuring that the standards are high are crucial. Not only that, the same tools can be used for any project that makes use of the framework. The fact that python gives the programmers a lot of freedom is both a good and risky. To avoid producing a framework that is hard to understand we relied on static analysis tools to enforce widespread standards of the python language.

Testing is also essential, as it provides a set of guarantees that the system works as expected, and simplifies refactoring.

5.2.1 Static Analysis

The static analysis employed is based on basic linting, style enforcement, and static typing. For the linting part, pylint is used, so that any potential error-provoking code is caught before it can cause any damage. The style enforcement is done with flake8, which is based on the community's python enhancement proposals (PEPs) and thus reflects the de-facto standards in python coding style. Finally, the static

Name	Statements	Miss	Cover
localization\base_actor.py	13	0	100%
localization\localization_service.py	138	82	41%
localization\locator.py	29	5	83%
localization\math_utils.py	43	6	86%
localization\messages.py	25	3	88%
localization\observed_item.py	57	25	56%
localization\particle_filtering.py	14	1	93%
localization\view.py	68	45	34%
tests\localization_service_test.py	22	0	100%
tests\locator_test.py	14	0	100%
tests\math_util_test.py	27	0	100%
tests\observed_item_test.py	16	0	100%
tests\particle_filtering_test.py	7	2	71%

Table 5.1: Code coverage.

typing is done via mypy, which provides a custom syntax for specifying the types of variables, parameters, return types, etc.

These tools need to be run each time the code is changed to ensure its correctness, but if that is not done, the actual execution of the program is not hindered by the presence of problems with the static analysis. To improve this aspect, the static analysis is executed as part of a continuous integration process using GitLab's CI tools.

5.2.2 Testing

Pytest is the testing framework in use for the project, as it provides a fairly widespread and simple approach to test writing, together with automation tools to be used in the previously described continuous integration processes. The tests written for the framework are mainly concerned with the aspects that need to be extended by the users. An overview of the coverage is presented in Table 5.1.

Chapter 6

Conclusions

The thesis' main concern is about the design and implementation of a framework that can aid in the development of indoor localization systems based on the angle-of-arrival. Nevertheless, the path to achieve such a result is filled with other challenges and research topics that have been discussed and dealt with. Thus, the thesis provides various contributions to the scientific community.

First and foremost, the development of a framework that is both a tool for companies to fine tune their products, and for researchers. The academia can benefit from this framework by using it to validate the performance of new telecommunication technologies that can be used for indoor localization, using it as a benchmarking tool. Another important contribution is that someone developing a new data fusion algorithm could easily test it in a variety of scenarios with minimum effort, possibly even using real world data from other researchers so that it is tested against specific use-cases.

Another important aspect, mostly discussed in Chapter 2, is an overview of the localization problem using angle-of-arrival. Even though this is not a new topic itself, most surveys just treat it as a small part of the whole indoor-localization landscape, and while that is not completely false, the new Bluetooth specification made it one of the most promising enabling technologies.

Finally, this thesis is, to the best of our knowledge, the first work to actually implement an indoor localization system based on the novel Bluetooth 5.1 specification. The hardware in use is development hardware not suited for commercial use, and the tested scenarios are very limited as well, so this is just the first step towards understanding and approaching the development of this kind of system.

The achievements of this thesis are all promising starting points for new researches and for improving the currently obtained results. Some of the most immediate and important paths that should be taken from this point on are presented in next section.

6.1 Future works

Among the possible future works that stem from the research done, the ones concerning improvements of the framework are discussed the most, as they are deeply connected with the main point of the thesis.

The first and most immediate action to be taken that would improve the quality of the proposed work, is the gathering of new experimental data using the presented situated system. This time, it would be necessary to provide a more stable means of moving the ObservedItem, a room that is fully isolated from electromagnetic interferences, and a broader range of tests. The additional tests need to be done in larger areas compared to the few centimeters used here, possibly adding a larger set of Locators which would provide interesting insight on the actual improvements that additional Locators can bring to the system. Experimenting with different hardware could also be interesting, but the main focus remains on repeating the same experiments in controlled scenarios.

Implementing different data fusion techniques is another important improvement that would give the system another important feature, as it might sometimes be useful to perform data fusion using different techniques, without the burden of implementing them from the ground up. HMM and Kalman filters are just some of the most important ones that could be the subject of this enhancement, but also more sophisticated, ad-hoc, or neural-network-based techniques are eligible for improving this side of the framework. While particle filtering is the best suited algorithm due to the problem's characteristics discussed in Chapter 3, each scenario and use case has its peculiarities and should be dealt with in the most appropriate way.

Finally, it is important to talk about GUI features that would enable the use of the framework as a fully-fledged user application with the aid of few extra steps. Most of the setup that needs to be done could be moved to the graphic interface, so that an end user could choose the number of ObservedItems and Locators, their positions, their movement over time, the dimensions of the room, and so on. Another key aspect of that would be tuning the parameters of the data fusion algorithm directly from the GUI. All of these could be seen just as minor inconveniences from a computer scientist's point of view, but if the tool needs to be used by someone with poor programming skills, the aid of a user interface would greatly reduce the chance of making mistakes.

The graphic interface could also be used for providing a standardized way of presenting and analysing the results of some batch testing, showing the average accuracy and precision based on the different values for each parameter.

Additional GUI developments would be concerned on commands to control the execution of the simulation, providing step, stop, and start functionality, configuring the speed of the displayed simulation, and the possibility to go backwards.

These options could be used by an end user to check some specific moments of time in which the system is not behaving as expected.

Bibliography

[1] Gul A. Agha. *ACTORS - a model of concurrent computation in distributed systems*. MIT Press series in artificial intelligence. MIT Press, 1990.

[2] Ahmed Yassin Al-Dubai, Youssef Nasser, Mariette Awad, Ran Liu, Chau Yuen, Ronald Raulefs, and Elias Aboutanios. Recent advances in indoor localization: A survey on theoretical approaches and applications. *IEEE Commun. Surv. Tutorials*, 19(2):1327–1346, 2017.

[3] C. BASRI and A. El Khadimi. Survey on indoor localization system and recent advances of wifi fingerprinting technique. In *2016 5th International Conference on Multimedia Computing and Systems (ICMCS)*, pages 253–259, 2016.

[4] Federico Bonafini, Dhiego Fernandes Carvalho, Alessandro Depari, Paolo Ferrari, Alessandra Flammini, Marco Pasetti, Stefano Rinaldi, and Emiliano Sisinni. Evaluating indoor and outdoor localization services for lorawan in smart city applications. In *2nd Workshop on Metrology for Industry 4.0 and IoT MetroInd4.0&IoT 2019, Naples, Italy, June 4-6, 2019*, pages 300–305. IEEE, 2019.

[5] Marco Cominelli, Paul Patras, and Francesco Gringoli. Dead on arrival: An empirical study of the bluetooth 5.1 positioning system. In Yiannis Pefkianakis and Kate Ching-Ju Lin, editors, *Proceedings of the 13th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization, WiNTECH@@MobiCom 2019, Los Cabos, Mexico, October 25, 2019*, pages 13–20. ACM, 2019.

[6] Khaldon Azzam Kordi, Abdulraqeb Alhammadi, Mardeni Roslee, Mohamad Yusoff Alias, and Qazwan Abdullah. A review on wireless emerging iot indoor localization. In Nur Idora Abdul Razak, Mohd Fais Bin Mansor, Nani Fadzlina Naim, and Wan Norsyafizan W. Muhamad, editors, *5th IEEE International Symposium on Telecommunication Technologies, ISTT 2020, Shah Alam, Malaysia, November 9-11, 2020*, pages 82–87. IEEE, 2020.

[7] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach, Third International Edition*. Pearson Education, 2010.

[8] Nitesh B. Suryavanshi, K. Viswavardhan Reddy, and Vishnu R. Chandrika. Direction finding capability in bluetooth 5.1 standard. In Navin Kumar and R. Venkatesha Prasad, editors, *Ubiquitous Communications and Network Computing*, pages 53–65, Cham, 2019. Springer International Publishing.

[9] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. *Artif. Intell.*, 128(1-2):99–141, 2001.

[10] Martin Woolley. Bluetooth direction finding: A technical overview. `https://www.bluetooth.com/bluetooth-resources/bluetooth-direction-finding/`. (Accessed: 11.03.2021).

[11] Faheem Zafari, Athanasios Gkelias, and Kin K. Leung. A survey of indoor localization systems and technologies. *IEEE Commun. Surv. Tutorials*, 21(3):2568–2599, 2019.

[12] Augustin Zídek, Shyam Tailor, and Robert Harle. Bellrock: Anonymous proximity beacons from personal devices. In *2018 IEEE International Conference on Pervasive Computing and Communications, PerCom 2018, Athens, Greece, March 19-23, 2018*, pages 1–10. IEEE Computer Society, 2018.