Swarm robot localization from noisy distance measurements

Electrical Engineering BSc. Thesis

Mitchel Chandi
M.V.Chandi@student.tudelft.nl
4230760

Jonas Carpay
J.M.Carpay@student.tudelft.nl
4169948

June 2017

Supervisor:
Chris Verhoeven

Technische Universiteit Delft

Abstract

In this thesis, we will describe the design process of a particle filter implementation for location estimation on the DeciZebro swarm robotics platform. We will discuss why we chose a particle filter as our filtering methods, what measurements have been taken to support its use, how it is implemented, and how we can test our implementation in the absence of the robot hardware.

Our results show that particle filtering is a promising approach that is especially suited for use on the Zebro. We also emphasize, however, that there are many variables that affect the particle filter's effectiveness, and that it is therefore difficult to draw conclusions about the final implementation.

Contents

1 Introduction

The *Zebro project* is a swarm robotics research group at the TU Delft. Its mission is "to design, plan and build self-deploying, fault-tolerant, inexpensive and extremely miniaturized robust autonomous roving robots to cooperate in swarms, capable of functioning on a wide spectrum of topology and environment that can quickly provide continuous desired information with the help of distributed sensor systems and carry and support payloads suitable for a wide range of missions.[1]" It started in September 2013, and has now been absorbed by the TU Delft Robotics Institute as part of their research endeavors into swarm robotics.

The idea of swarm robotics is inspired by nature, in particular by small insects and birds who exhibit the so-called *swarm behaviour*. As a group, a swarm can accomplish complex tasks, despite the fact they are not considered 'smart' individually. By all following the same simple set of rules, the swarm as a whole can exhibit *emergent behaviour*, behaviour that is not explicitly programmed into any one of its members. This idea that a group of simple individuals can deal with more complex tasks is the cornerstone of swarm robotics. Furthermore, an entire swarm is resilient, as every individual member is expendable. This means that swarm robotics has unique applications, such as in disaster areas that are inaccessible to humans or larger robots.

Currently, the Zebro project is working to redesign the robots to (further) improve producibility, reparability, stability, and cost. Being able to produce useful Zebro units on a large scale will provide fertile ground for swarm robotics research efforts. There are several different versions of the Zebro, each with a different form factor and design goal. Our efforts are focused on the *DeciZebro*, which is the size of an A4 sheet of paper and is designed to be extremely modular.

Our group's goal is the design of a communication and localization module for the DeciZebro. The module has two purposes. The first is to provide a Zebro robot with the positions of robots in its vicinity. The second is to facilitate wireless communication within the Zebro swarm.

We have decided on using range measurements for localization. This has led to the following subdivision of our module:

- The *ranging* submodule measures the distance between Zebro robots.
- The *communication* submodule sets up a decentralized mesh network for communication.
- The *processing* submodule takes the measurements provided by the other submodules and uses them to construct a local (i.e. in the Zebro's neighbourhood) model of the swarm.

1.1 Problem Definition

The topic of this thesis is the processing submodule. Our submodule has access to noisy distance measurements to other robots (courtesy of the ranging submodule), and a communications network (courtesy of the communications submodule), and we have to combine these to provide some model of the swarm. This is a difficult problem that is the subject of much research (see section 1.2). Our final product consists of the following deliverables:

Particle filter firmware We provide the code that combines measurements, and provides a convenient interface to access the filtered information. All software on the microcontroller, including this filtering code, is written in C.

Compass hardware In order to break rotational symmetry we will provide a compass module. Aside from our own localization algorithm, directional information is also useful for common swarm algorithms[2].

Simulation software There is no operational DeciZebro yet, and we will not be able to perform any tests until the other two submodules are functional. We will write software for simulating a Zebro swarm, with which we can test our localization algorithm.

1.2 State of the Art

The rise in popularity of UAV's, electric cars and other consumer robotics has fueled extensive research into localization techniques. The particular case of range-based localization has also been the subject of much research[3]–[5], even at the TU Delft[6]. Most of the work in robot localization, at least those parts that are relevant to our module, are specific cases of the indoor localization problem, which arises due to the limited effectiveness of GPS in indoor environments[7].

Every solution needs to address the finite accuracy of the sensors and the conflicting measurements resulting from them. In particular, distance measurements are susceptible to large inaccuracies due to the fact that they are based on the propagation of waves and thus have to deal with interference and environmental circumstances. The approaches can be divided into two categories; those that use analytic solutions, and those that use a probabilistic model (i.e. Bayesian inference). Analytical solutions (e.g. triangulation) like those proposed in [3]–[5] solve a system of equations that follow from the measurements, and then apply convex optimization (i.e. least squares methods) to minimize the error from conflicting measurements. Bayesian inference uses Bayes' rule to continually update a best estimate of the system state[8]. The two most common uses of Bayesian inference are the Kalman filter[6], [9], [10] and the Particle filter[11]–[14]. We will choose to use a particle filter, for reasons described in section 4.1.

Despite the large amount of literature on the subject, there are no solutions that we can directly copy for use on the Zebro. The design space is vast, as there are many variables that each affect what solutions are viable. Examples include the chosen sensoring method, the required accuracy, available processing power, indoors/outdoors, the required update frequency, and the availability of nodes with fixed known positions. There is also a large number of research papers for what seems to be a surprisingly small number of proven, real-world implementations of swarm robot localization. Still, there are many common principles behind the different implementations that have become the basis for our own design as well.

We will now discuss some successful implementations of swarm robotics localization and the methods they used.

1.2.1 Cricket localization system

The cricket system uses nodes that send RF pulses followed by an ultrasonic pulse.[15] The nodes have stationary known positions, in for example a room, and sends the pulses at a predefined frequency. Listeners in the room can localize using the TDOA (Time distance of arrival) between the received RF and Ultrasonic signal. The main advantage of this system is that it is decentralized; every mobile listener in the area can autonomously locate itself without the need of a central administrator which keeps track of the individual users. One significant disadvantage is that it needs stationary beacons.

1.2.2 Kilobot

The Kilobot is a low cost swarm robot developed at Harvard University. They are able to execute commands in large groups, up to a thousands. The Kilobot uses triangulation to determine its relative position in the swarm [16]. The triangulation method requires the distance measurement from 3 neighboring robots which know its own location in a triangle formation. From these relative distances, the fourth robot can determine its location in the formation. This idea is shown graphically in figure 2.1. This analytical method is easy to understand and widely used. It does pose some challenges, however. Communication is essential for this to work, and it should also be kept in mind that the time between distance measurements should be kept as low as possible to improve accuracy. Moreover, the three surrounding robots should know their own position in the environment for this to work. Otherwise, no meaningful statement can be made based on the distance measurements only.

1.3 Synopsis

We start by outlining the program of requirements to our module in chapter 2. This is split into two parts; section 2.1 gives the requirements pertaining to the module in general, while section 2.2 describes the requirements for the processing submodule specifically.

Chapters 3 and 4 describe the design process. They are also split between the design of the module (chapter 3) and the processing submodule (chapter 4).

Chapter 5 will describe the implementation of our submodule, and in chapter 6 we will conclude with a discussion of the results.

2 Program of Requirements

This program of requirements is divided into two sections. Section 2.1 consists of the technical specifications as given by the Zebro group. These apply to our module as a whole, and they motivate the module-level design. In section 2.2 we will discuss requirements specific to our submodule.

2.1 Module Requirements

This set of requirements was drawn up in collaboration with the Zebro group. They mostly consist of hardware specifications. Because our deliverables consist primarily of software and firmware, only a few apply directly to our submodule. Instead, they apply to the overall module design (chapter 3) and the ranging submodule[17] and communication submodule[18].

Functional requirements describe the basic functionality expected of the localization and communication module, system requirements describe the technical specifications of the final hardware, performance requirements define the required accuracies on the produced data, and development requirements describe the constraints imposed by the manufacturing process.

2.1.1 Functional Requirements

[**F-1**] The module should be able to locate multiple neighbouring Zebros

- [**F-1.1**] The module should be able to be used on and locate charging stations and similar units
- [**F-1.2**] The module should be able to distinguish different users of the module

[**F-2**] The module should be able to receive and transmit packets to neighbours

- [**F-2.1**] The module should be able to make contact with at least 3 neighbouring Zebros
- [**F-2.2**] The module should be able to broadcast housekeeping data for inspection by users

2.1.2 System Requirements

[**S-1**] The mechanical, power, and digital connections must conform to the specifications outlined in [19]

[**S-2**] The module must conform to the Zebrobus protocol outlined in [20]

[**S-3**] The module must respect the modularity of Zebro modules

- [**S-3.1**] The module should be a slave to the Zebro bus and cannot make requests itself
- [**S-3.2**] Use of the module cannot inhibit the functionality of other modules on the Zebro
- [**S-3.3**] With the exception of data broadcast on the Zebro bus, the module should collect data exclusively using its own sensors
- [**S-3.4**] The module should communicate its data to the Zebro by responding to requests over the Zebro bus
- [**S-3.5**] The modules should be interchangeable between robots and easily replaceable

[**S-4**] The module should be able to be used in indoor environments

[**S-5**] The communications network should be homogeneous

- [**S-5.1**] Communications cannot rely on a predetermined master node or network hub
- [**S-5.2**] An individual Zebro within range should be able to join the communications network at any time
- [**S-5.3**] Any individual Zebro in the network should be able to leave without affecting network functionality

[**S-6**] The module should be usable on stationary elements, such as a charging station, without any changes to the hardware

[**S-7**] The module connects to the Zebro power system (see [**S-1**])

[**S-7.1**] The power system is rated at 13 V to 19 V[1]

[**S-8**] The module should be able to provide debugging information over the Zebro bus

2.1.3 Performance Requirements

[**P-1**] The module should be able to locate other Zebros with an accuracy of $<10\,\%$

[**P-2**] The module should be able to locate Zebros within a range of at least 10 m

2.1.4 Development Requirements

[**D-1**] The module production cost shall not exceed €150 per unit

[**D-2**] The module PCB should be able to be fabricated using a pick-and-place service

2.2 Processing Submodule Requirements

We will now formulate the design criteria for our submodule. These will drive the design decisions in chapter 4.

2.2.1 Inherited Requirements

Of the requirements listed in chapter 2, only a few directly apply to our submodule:

- [**F-1**] states that we need to distinguish between different units. This is accomplished by having every unit include its own ID when sending a beacon, and keeping separate models for every beacon. We will discuss the beacon in section 4.3.
- [**S-3.2**] implies that our approach cannot restrict the Zebro's movement, and as such has to work for all possible paths the Zebro can take. This is a stricter version of [**S-6**], which requires the approach we take to still work on units that do not move.
- [**S-8**] requires us to provide debug information where possible.
- [**P-1**] states that we need at least 10% accuracy.

2.2.2 Design Requirements

We will now present a list of issues that need to be addressed by any solution. These criteria are problems that are either inherent to the problem or the Zebro, or arise from design decision made in other parts of the module. These are supplementary to the requirements stated above.

[**d-1**] Our module is still in development, and the design is subject to change. Over the course of our project, any of the submodules may find that their approach is not viable. For example, should TDOA methods prove to be too difficult to implement, we might consider switching to signal strength measurements for ranging. Another example is that we will not be able to empirically determine the accuracy of our ranging method until we have a working prototype. It is preferable for us to find a solution that is general enough to work for different approaches. The more general the solution we present, the more adaptable it will be to future designs for localization modules.

[1] There are no specifications yet for the maximum drawn power/current

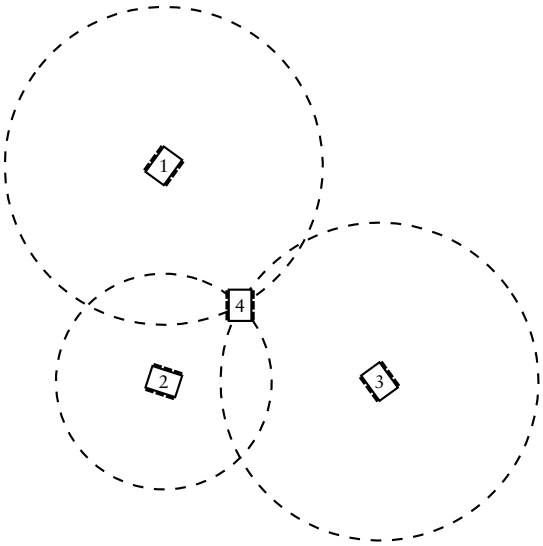


Figure 2.1: We can use the distances measured by robots 1, 2, and 3 to locate robot 4, but only if we know their positions. If we do not know their positions, any rotated version of this diagram has the same distances between robots and is therefore equally valid.

[d-2] There is no functional DeciZebro yet. The hardware design is largely complete and there are specifications for its interfaces (see [19]), but we cannot perform real-world tests and should aim to keep our designs as general as possible.

[d-3] Measurements are noisy. Our solution will have to be able to account for some degree of noise on distance measurements. In order to account for this, we expect the ranging team to provide an accurate characterization of the noise.

[d-4] Communications are random. Because of the network homogeneity communications are not centrally scheduled. The scheduling algorithm used by the communications submodule does not make any reliable guarantees about when messages are broadcast[18].

[d-5] If the Zebro unit were able to perfectly measure distances to surrounding units, share those measurements with other units instantaneously, we would still have rotational symmetry. The construction of a map from distances between points is called *multidimensional scaling*, and is solved by simple linear algebra. The problem is that it is impossible to recover the original orientation of the map. Our solution needs a way to break this rotational symmetry. You can see in figure 2.1 how we can derive a position from measurements only when we know the origins of the distance measurements.

[d-6] There are many interpretations of a sequence of distance measurements. Figure 2.2 shows how there are multiple possible hypotheses given two sequential distance measurements. Additional information is needed to distinguish between the paths taken.

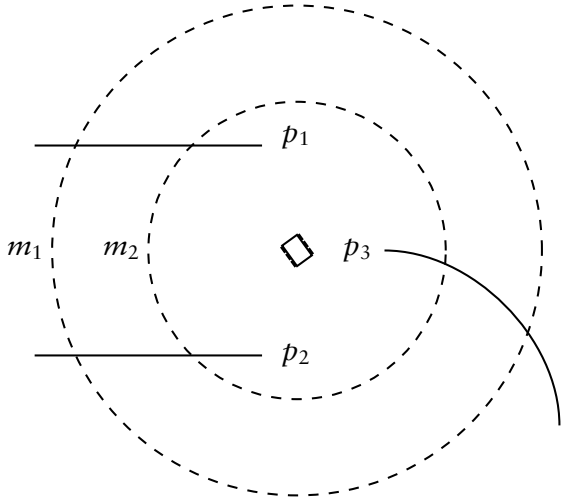


Figure 2.2: Distance measurements alone are not sufficient to make any meaningful conclusions about a robot's path. As you can see, p_1, p_2, and p_3 can be explained by the sequence of distance measurements $\{m_1, m_2\}$. We need more information to distinguish between these paths. Note that knowing the shape of the path is not necessarily sufficient in the absence of other information, as evidenced by the fact that p_1 and p_2 have the same shape.

3 Module Design

The module subdivision is a result of the design decision we made at the top-level. In this chapter we will discuss those decisions, and in chapter 4 we will discuss the design of the processing submodule. As defined in section 2.1, the module has specific requirements to uphold. We will clarify the relevant design choices for Ranging Method in section 3.1 and in section 3.2 for Processing.

3.1 Ranging

The Zebro needs to be able to estimate its position relative to its neighbours. We identified a number of approaches:

GPS GPS cannot be used because this conflicts with the **[S-3.2]**, namely the use of the satellites which GPS uses. **[S-4]** also states that the system must be able to function indoor, which GPS cannot do (at least not accurately) [7].

RSSI [21] shows two problems with using RSSI as a ranging technique for bluetooth devices: the RSSI is not stable since the Bluetooth protocol takes measures to ensure the highest quality link and the RSSI is dependent on device orientation. [22], [23] gives values for the orientation dependency: in worst case the result for a measurement done at 1 meter distance gives RSSI values which correspond to 10 meter. Using RSSI for ranging might be possible in static environments with fixes device orientation but is not adequate in a system with moving devices.

Angle of Arrival [24] shows that it is possible to determine the angle of arrival (AOA) using rotating directional antenna's, antenna arrays or beam forming antenna. This (AOA) can be used to map devices and estimate the distance between these mapped devices. The problem is that these systems are big, expensive and complex making them unfavourable for use on a Zebro and within the time frame of the BAP.

Acoustic [25] uses acoustic signals to determine time-of-arrival (TOA). Two-way sensing is used to determine the distance without the need for synchronised clocks. The result is a low cost, high accuracy device which operates in the audible sound spectrum. The range for indoor applications is roughly 5 meters due to reflection/multipath.

Infrared [26] describes a system of miniature robots which use IR to determine the distance between robots. It highlights that the range of this system is only 3 meters, prohibiting its use on the Zebro.

Ultrasonic with RF pulse [15], [27], [28] describe the work done on a system that uses ultrasound in combination with the RF communication channel to determine distance between two devices. The system measures the *time difference of arrival* (TDOA) of a RF transmitted ID tag and an ultrasonic pulse. This time difference is related to the distance and gives results with an accuracy of 1 cm with proper filtering techniques. This system uses beacons with a fixed position to determine the relative distance of moving devices. The range of the system is roughly 10 meters.

After researching and comparing the different techniques, ultrasound has the most potential. Both the accuracy and the range are sufficient. Other researched systems fall short on one of these fronts. Bluetooth systems have a high range but poor accuracy for moving devices. Acoustic and IR systems have a high accuracy but the range is limited.

3.2 Communication submodule

For the communication part a few requirements are set for the communication interface of the zebro: nearby zebros must be able to independently communicate with each other without the use of routers and gateways. In other words, the network must be autonomous and homogeneous. This implies that the network should be able to run without the need of some master/controller devices such as routers to control it. Furthermore all the nodes in said network are to be considered as equal. This type of network is called a mesh network, where no predefined routes are used to communicate. Possible solutions for this problem are:

Bluetooth Bluetooth has proven over the last two decades to serve well in simple applications where two devices devices need to connect with each other with easy coupling and low data rates. It is however possible to set up Ad-hoc networks with Bluetooth. [29], [30] At the moment there are no exact specifications for Bluetooth enabled mesh, but there is a large community trying to develop these, resulting in a few time-efficient algorithms such as BlueStars, BlueMesh, and BlueMIS that seem fast enough for combined ranging and communication. [31]

WiFi Another very commonly used communication protocol is Wi-Fi. Just like Bluetooth, Wi-Fi makes use of the IEEE802.11 standard. Wi-Fi suits better than Bluetooth in applications where high bandwidth is desirable and some degree of client configuration. With WIFI the transmission rate is inversely proportional to transmission range which also corresponds to the transmit power level. [32]

XBee ZigBee is a widely used specification for wireless communication networks [33]. Just like Wi-Fi and Bluetooth it can work in various bands, from low RF bands (868 MHz) as well as the 2.4 GHz bands. Most implementations with ZigBees use a point-to-point topology [34] but ZigBee also defines a mesh network topology called ZigBee Mesh [35], [36]. This however is a mesh implementation that requires a coordinator to handle routing and maintain proper setup of the network. An alternative to ZigBee Mesh is DigiMesh. This is a mesh topology using Zigbee but with homogenous nodes and no need for a coordinator [37]. An example of an implementation with DigiMesh is [38].

As there are no working implementations yet apart from DigiMesh, which we can use as solution for the communication part of our module, the choice is clear. The communication will be done via DigiMesh as this is a proven concept.

3.3 Processing

The final step is the processing of all information. We need to create a model of the environment for every Zebro. This means that we need an algorithm to translate the sensor data of Zebros into updates of our localization model. Most localization problems involve a stationary anchor point or some reference point which is used for the localization. Moreover, methods such as triangulation are not adequate since this poses problems which are discussed in 4.1.1. Therefore, the decision was made in an early stage to divide our module into three submodules (as mentioned before):

- Ranging
- Communication
- Processing, which is the subject of this thesis

4 Processing Submodule Design

This chapter describes the design process of the processing submodule. Its organization mirrors the order in which these issues came up when designing the processing submodule.

4.1 Filtering Method

Distilling noisy measurements into an estimate of the state of some system is called *filtering* (see the discussion on nomenclature in section 4.1.5). As described in section 1.2, there are three general approaches that come up in literature; analytical methods, Kalman filtering, and Monte Carlo methods/particle filtering.

4.1.1 Analytical Methods

Analytical methods are arguably the most straightforward. They solve a system of equations to obtain estimates for the positions of the robots. Those estimates are then typically consolidated using convex optimization (i.e. least squares methods). Triangulation and multidimensional scaling are examples of analytical methods. Compared to other methods, analytical methods hold little state, i.e. they keep track of a number of past measurements, but other than that, they use little memory. We will base our decision on the extensive discussion of analytical methods in [5].

The authors describe two situations. In the first, units are allowed to move freely. This suffers from the problem of indistinguishable paths (these paths are called *degenerate*). This is a fundamental problem in the absence of other information: "…there are motion patterns for which any algorithm that does not enforce a very strict motion coordination … cannot recover the relative pose of neighboring robots." The second approach solves this problem by using "a distributed localization algorithm that uses a simple stop/move motion coordinate scheme, and requires communication proportional to the number of neighbors."

The main advantage of these methods is that they are relatively cheap, both memory- and computation-wise. They have also been proven to work, as they are used in the Kilobot swarm robotics platform[16], specifically the version with coordinated movement. Unfortunately, the fact that they either have ambiguities that would not be present with the other approaches, or require coordinated movement that is in violation of **[S-3.2]**, are significant disadvantages.

For these reasons, we will not use analytical methods. The fact that they have been proven to work on the Kilobot platform and their low computational complexity mean they are effective means of swarm robot localization. Due to the reasons outlined above, however, they are unsuitable for the Zebro.

4.1.2 Bayesian Inference

Bayesian inference provides us with a framework for dealing with imperfect information in a mathematically sound way. The cornerstone of Bayesian inference is *Bayes' rule*:

$$P(H|E) = \frac{P(H)\ P(E|H)}{P(E)} \qquad \text{(Bayes' rule)}$$

In Bayesian inference, it is interpreted as giving us the probability of some hypothesis H (e.g. "Robot 2 is 2 m in front of robot 1") being true given the occurrence of some event E (e.g. "Robot 1 detects robot 2's signal at a distance of 4 m"). We start with an initial probability $P(H)$, called the *prior*, which can initially be just a guess. If we know the distributions $P(E)$ and $P(E|H)$, and that some event E has occurred, we now have a new probability $P(H|E)$, called the *posterior*. Note that the prior and posterior are only defined relative to some event E. When mentioning the prior, we will consider it to include all events that have occurred before E. When talking about the initial guess for the prior in the absence of any data, we will refer to it as the *initial distribution*.

There are two common filtering methods that rely on Bayesian inference; the *Kalman filter*[39], and *Monte Carlo localization*, or *particle filter*[11], [12]. The algorithms for both are similar, and follow directly from Bayes' rule. They both have a state that represents the probability distributions of some vector of variables. When new information presents itself, that state is scrutinized using the new information, and then updated. In between measurements, the state is updated according to some model of the time-behaviour of the state. There are more ways to use Bayesian inference, such as the grid-based method that is briefly discussed in section 4.1.4, but these are the two that are commonly mentioned in literature[8].

4.1.3 The Kalman Filter

In a Kalman filter, the state vector of a system is represented as a vector of *Gaussian* random variables, i.e. a vector of their mean values, and a covariance matrix that captures the interdependences between these variables. This is in contrast to ordinary state-space models, in which variables are considered non-random. When performing an update due to some new evidence, we model our inputs again as a Gaussian vector, where the vector of means is the control vector, and the covariance matrix the control matrix. Using these, we update the mean vector and covariance matrix to get a new posterior distribution. A more extensive discussion of the mathematics behind the Kalman filter can be found in [8], [39].

The Kalman filter has computational cost, is guaranteed to be optimal (i.e. minimizes least squares error) for Gaussian noise, is easy to implement and scientific computing packages such as Matlab or NumPy all have good Kalman filter libraries[40]. It is a popular solution to the filtering problem and a good fit for our purposes, as evidenced by its use in [6], [9], [10]. Arguably the main disadvantage is that it only works on signals with Gaussian noise, but this can be addressed by one of the many variations on the Kalman filter discussed below.

One of these variations is the popular *Extended Kalman Filter*, or EKF[9]. The extended Kalman filter is a common way of adapting the Kalman filter for variables that are nonlinear. In it, nonlinear variables and their statistical covariances are replaced by those of their first-order Taylor series approximations.

If the first-order approximation is not sufficient, one can consider the *Unscented Kalman Filter*, or UKF. The UKF divides a single probability distribution into multiple Gaussian random variables. These variables "completely capture the true mean and covariance of the [distribution], and when propagated through the true nonlinear system, captures the posterior mean and covariance accurately to the 3^{rd} order (Taylor series expansion) for any nonlinearity.[41]"

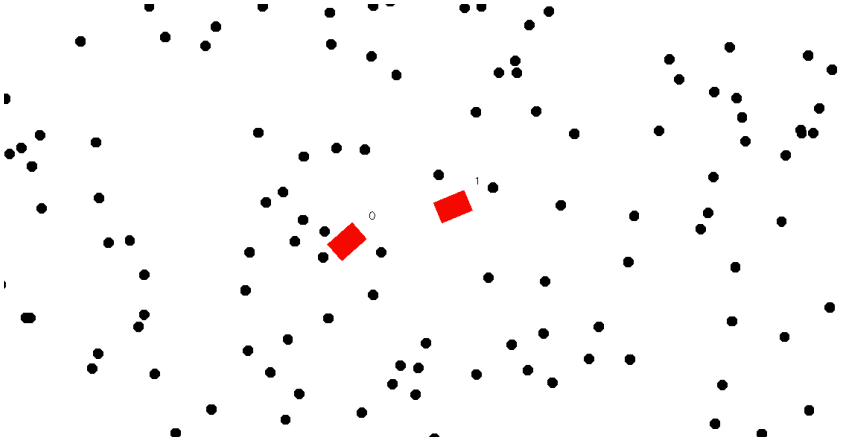
4.1.4 Grid-Based Methods

One straightforward way to employ Bayesian inference for localization is to divide space into a grid[12], [14]. Say you are interested in tracking the position of some object o. In our case o is a neighbouring Zebro. If you assign each cell in the grid a probability that o is in that cell, you have a grid that approximates the probability distribution of the position of o. Once a new piece of information presents itself, we can use Bayes' rule to update the probabilities in the grid cells. The two main advantages over the Kalman filter are its easy implementation and that it makes no preference for variables that follow a Gaussian distribution, as a scalar grid is able to approximate any probability distribution.

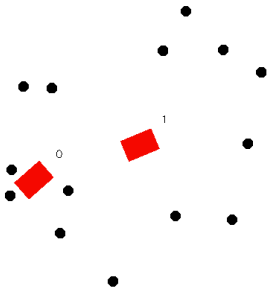
The key problem of this approach, and likely the reason that we were unable to find any uses in practice, is that of *degeneracy*. Degeneracy, in this context, refers to the fact that as we get more certainty about o's position, the probabilities of all but a few cells will approach 0. This means that we spend processing power on grid cells whose probability is virtually zero in both the prior and posterior, and lose accuracy to the quantization imposed by the grid. Both of these issues are addressed by the particle filter.

4.1.5 The Particle Filter

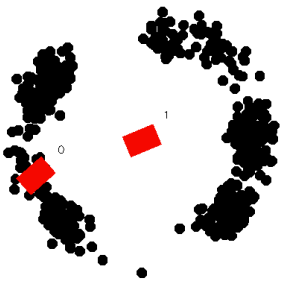
Instead of assigning the probabilities to cells in a grid, we make a list of explicit points in space to track, called *particles* and assign probabilities, called weights, to them. When the particles' positions lie on grid points a particle filter is equivalent to a grid-based Bayesian method. What makes a particle filter fundamentally different is that as degeneracy increases, it performs *resampling*. There are many different ways to perform resampling[42], but the common idea is to update the particles' positions such that their assigned probabilities are more evenly distributed.



(a) Initially particles are randomly distributed, and each is assigned the same weight.



(b) Once weights converge on a few particles, we perform resampling. Note that the number of particles stays the same, but resampling reduces the number of distinct positions.



(c) The motion model dissipates the particles again, in this case they are normally distributed around their initial positions.

Figure 4.1: An example of resampling and motion model application

Figure 4.1 shows a resampling/motion cycle. In the initial configuration no information is present and all positions are equally likely. This manifests as a particle cloud that is randomly and evenly distributed, with all particles having the same weight. When measurements have been made such that the nonzero weights converge on a small number of particles, we resample the particles. This is done by drawing a new set of particles (the posterior set) in which each particle gets a position by randomly selecting a particle in the old set (the prior set) and copying its position. The probability that a particle in the prior set is selected is proportional to its weight.

A *motion model* describes the movement of the tracked object. It is a function that takes the elapsed time and original position of the object, and gives a probability distribution of the new position of the object. In between measurements, the motion model is used to update the particles to new positions. In figure 4.1, the tracked object stays still, and the motion model normally distributes the x- and y-positions around their original value. Figure 4.1c specifically shows the dissipation due to this simple motion model.

Like the grid-based method described above, the particle filter makes no preference for Gaussian distributions. As long as we are able to formulate a probability density function for a piece of information, we can use it to assign weights to our particles and hence update our state estimation. This powerful property makes it easy to do cooperative localization, as we will see in chapter 5.

Nomenclature

'Particle filter' and 'Monte Carlo localization' are just two of the names found in literature that refer to the same concept. The name 'particle filter' comes from the fact that this method, like the Kalman filter, describe approaches to the *filtering problem*. The filtering problem is the general problem of making a best guess of the state of some system of which we only have information that is inaccurate, indirect, or both. 'Particle' refers to our state estimation, which consists of particles whose density approximate our probability distribution. The Monte Carlo names show that this method is a member of the general class of Monte Carlo methods, that all have in common that they rely on random sampling. The most common name seems to be 'particle filter', which we will therefore use for the rest of our thesis.

4.1.6 Comparing the Kalman Filter and Particle Filter

Having decided that analytical solutions are not suitable for our problem (see section 4.1.1), our choice is between the Kalman filter and particle filter. The trade-off between these methods is one between computational cost and flexibility.

A Kalman filter is much cheaper computationally, but at the cost of being much more rigid. It requires the user to formulate and tweak the control matrices of their information sources, which gets more difficult as the system gets more elaborate. The particle filter on the other hand, is very flexible but computationally expensive.

Kalman filters seem to be more prevalent in literature. We think this is because the computational power can be scarce on mobile robots, which makes the particle filter less suitable than the computationally cheap Kalman filter. **[d-1]**, **[d-2]** and the Zebro's large battery make our situation different, however.

The Zebro is designed to be a versatile modular platform, and as such needs a battery that powers a larger range of applications than e.g. the Kilobot's coin cell battery[16]. [43] contains a particle filter implementation that runs on an Arduino, and an Arduino can run for a day on a 9 V battery, so we will assume that a microcontroller running a particle filter will draw a negligible amount of power from the battery. Furthermore, the fact that neither our module nor the Zebro are functional yet requires us to keep our designs as flexible as possible.

These circumstances make a particle filter the obvious choice for a filtering method. Should it turn out that a particle filter is too demanding or too slow, we can consider implementing the filter on an FPGA device, which [44] describes as reducing the computational burden while achieving twelvefold speedups. An additional benefit of the particle filter is that it makes conversions between Cartesian and polar coordinates easy, which turns out to be very useful.

4.1.7 Resampling Algorithms

There is a number of approaches to resampling for particle filters[13], [42]. The main choice is between *Sampling Importance Resampling* (SIR) and *Sampling Importance Sampling* (SIS). Using SIS, we maintain a weight variable for every particle. Once the number of effective particles N_{eff} has dropped below some threshold, we resample them. In SIR, no weights are stored for the particles, instead using only the density distribution of particles to represent the probability distribution. Every time an event occurs, we use the associated probability density function to weigh our particles and immediately resample them.

While SIS is the most straightforward from a Bayesian inference point of view, we mostly see SIR being used in literature. We assume that this is because SIR has more predictable performance, since resampling is *always* performed, and because its implementation is more straightforward. An additional benefit of SIR is that not keeping weights means we do not need to normalize our probability distribution so that the sum of weights is always one.

4.1.8 Optimality and Accuracy

[P-1] requires us to provide an accuracy of at least 10 % on our final results. The Kalman filter has the optimality property[40], which means that for Gaussian noise sources, it will provide the best least-squares estimate given the data. This means that when using a Kalman filter, the output will be the most accurate it can be given the inputs, or in other words, the quality of our results is completely outside of our control as long as we use a Kalman filter.

While a Particle filter is based on random processes, and is therefore not optimal, we did not find any reason in the literature to think that it is significantly less accurate than a Kalman filter. We will therefore assume that **[P-1]** is outside of our control, as it is determined by the accuracy of the sensors. In order to give the end user an indication of the accuracy of the results, we will provide variances for all estimates produced by the filter.

4.2 Locomotion Estimation

As explained in **[d-5]**, rotational symmetry is a fundamental issue when reconstructing a map from distance measurements alone. When using a particle filter, this issue manifests when defining the motion model (see figure 4.2). The motion model gives a probability distribution for the Zebro given an initial position and a time delta. In the absence of any directional information, this distribution will also suffer from rotational symmetry around the origin.

Resolving this issue by using direction-sensitive ranging is both impractical and too expensive[17]. Instead, we will measure the compass bearing θ of the Zebro, and incorporate that information into the motion model. The signal that controls the movement of the Zebro's legs is broadcast on the Zebrobus, which we can use to estimate $|v|$, the speed of the unit. These two pieces of information will allow us to formulate an estimate of the Zebro's velocity v, and integrating v gives us an estimate of the displacement Δx. While neither of these information sources is likely to be very accurate, a particle filter only requires a probability distribution of the actual motion, the formulation of which allows us to account for inaccuracies in the data.

4.2.1 Compass sensor

The digital compasses, or magnetometers, to choose from are vary wildly in price. The price mostly seems to affect the accuracy of these sensors. Fortunately, we can easily account for inaccuracies in the sensor data by careful construction of our motion model. The cheapest category of magnetometers already have an accuracy of 2 degrees which is sufficient for our intends and purposes.

The HMC5983, for example, boasts an accuracy of 1 to 2 degrees[45] while being priced at 5.99 Euros with breakout board. This was the obvious choice, as it is a widely used magnetometer with plenty of available software libraries.

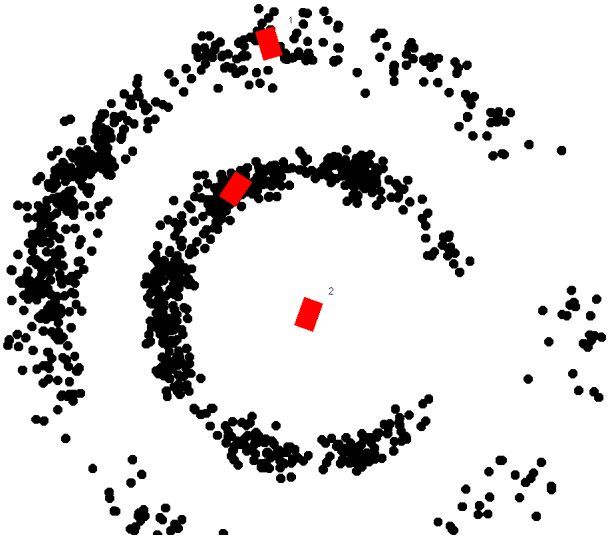


Figure 4.2: Rotational symmetry occurs when the motion model does not contain any directional information. Here, we see that the particles collect in rings, meaning that any position on the ring is equally likely.

4.3 Beacon Design

The TDOA ranging method we use performs localization by sending out distinct ultrasonic and RF pulses. When sending these pulses, we can include data into the RF pulse. We call these pulses and data a *beacon*. We will now describe the data that we include in a beacon. The implementation on the final hardware will include likely only include a subset of this data because of limited bandwidth. In our simulation however, bandwidth is unlimited, so we can experiment with what data is needed for the filter to properly converge, and under what circumstances.

Distance The distance, calculated from the TDOA.

Sender ID The ID of the sender. Every robot in the swarm has a unique ID.

Model The sender's internal model, containing its knowledge of the swarm.

Bearing The sender's compass reading.

5 Implementation

As explained in section 2.2.2, there is no functional Zebro yet, and our own module is still in development. This means that we will not be able to perform any real-world tests until the other two submodules have produced functional prototypes. In order to test our localization approach, we will write simulation software. This simulation is discussed in section 5.1. We will also discuss the implementation of the particle filter. This is described in section 5.2. Finally, in section 5.3 we will discuss our cooperative localization scheme.

Source Code

All code can be found in the Zebro git repository[1]. The firmware for the microcontroller is written in C, and the simulation is written using Haskell. All code in this thesis, like the data types in table 5.2, are given in pseudocode that mimics the Haskell implementation. It is not very fast, but its high abstraction makes it easy to test ideas. The C code is equivalent, but better optimized and much faster. What we call a *data type* in this thesis is a generalization of what is called a struct in C or an object in most object oriented languages.

Running the simulation is easy; download the Haskell Stack build tool, and run `stack build && stack exec sim` in the simulation's root directory.

5.1 Simulation

The simulation consists of a number of simulated robot units, each with an implementation mostly equivalent to that in the final firmware implementation. The simulation takes a number of parameters. These are described in table 5.1. Most of the discussion of our results in chapter 6 will be based on this simulation.

5.1.1 Simulating Distance Measurement Noise

Whenever a beacon is sent, we draw a number from a normal distribution with $\mu = 1$ and $\sigma = 0.1$. We multiply all reported distances by this number. Early reports from the ranging team indicate that this is more noise than will be present on the final distance measurements. Once they have good results on the noise characterization on the distance measurements we will replace this by a more accurate model.

5.1.2 Lennard-Jones Potential

One difference with the final implementation is that we now have control over the Zebro's movement, whereas on the actual implementation we can only provide information to the motion controller. In the simulation, a Zebro has two options of movement; it either moves in a straight line with some fixed velocity (possibly zero), or it moves according to its Lennard-Jones potential.

The Lennard-Jones potential is a simple model for atomic potential in which atoms form a crystal structure. The potential is given by

$$V_{lj} = \left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6} \qquad \text{(Lennard-Jones Potential)}$$

The first term gives an attraction when r, the distance, is larger than the desired distance σ. The second term causes a repulsion when $r < \sigma$. We calculate the velocity vector by first taking the derivative of the potential to get a force vector, and then integrating this force vector over all particles for every neighbour.

Due to the reasons described in section 4.2, localization requires the Zebros to move. Simply moving in a straight line can result in mirror symmetry, especially without cooperative localization, so we would like a movement scheme that has more interesting movement. The reason we decided to include Lennard-Jones movement is that it provides a good sanity check for our code. The Zebros should arrange themselves in a grid only if they manage to deduce the positions of the other Zebros to a reasonable accuracy.

[1] `https://svn-mede.ewi.tudelft.nl/git/zebro.git`

Table 5.1: Simulation parameters. The parameters above the divider are only relevant to the simulation, those under the divider are also configuration parameters on the final hardware.

`pixelsPerMeter`	The number of pixels per meter in the graphical output. This is essentially the zoom level
`timeScale`	A dimensionless factor with which all time deltas are multiplied. Used to speed up the simulation
`seed`	The seed for the random number generator. Having a fixed seed allows us to make the simulation deterministic while still being able to see the effect of randomness.
`beaconFreq`	Controls the frequency with which beacons are transmitted. This value controls the frequency in the entire swarm, not per robot. It is given in Hertz.
`nrOfParticles`	The number of particles that every Zebro uses to track other Zebros.
`s_lj`	The σ in the Lennard-Jones potential. See section 5.1.2 for details.
`distanceVar`	A factor that determines how much distance introduces variance in the motion model. Increasing this value means that more variance in the final position is introduced, the larger the time between two emitted beacons.
`timeVar`	A factor that determines the contribution of elapsed time to the motion model variance.
`constantProbability`	A constant that is added to all probabilities. Increasing this makes the particle filter more flexible at the cost of slower convergence.

Table 5.2: Data types

(a) The Zebro data type. This is Zebro *i*'s internal state as far as the processing submodule is concerned.

Field	Type	
i	$\mathbb{Z}$	The ID for this robot
Δx_{ii}	$\mathbb{R} \times \mathbb{R}$	The Zebro's displacement vector since its last emitted beacon
Model	{RobotModel}	A set of RobotModels (see table 5.3b)

(b) The RobotModel data type for some other Zebro *j*. This represents *i*'s knowledge about *j*.

Field	Type	
j	$\mathbb{Z}$	The ID for the tracked robot
Particles	$\{\mathbb{R} \times \mathbb{R}\}$	A vector of particles, of some constant length
Δx_{ij}	$\mathbb{R} \times \mathbb{R}$	Zebro *i*'s displacement vector since the last received beacon from *j*. This is used to calculate relative displacements.

(c) The Beacon data type. This is a beacon, sent by *j* and received by *i*.

Field	Type	
d_{ij}	$\mathbb{R}$	The distance between i and j. This is derived from the TDOA.
j	$\mathbb{Z}$	The ID of the sender
Δx_{jj}	$\mathbb{R} \times \mathbb{R}$	j's displacement since its last transmitted beacon.
SimpleModel	{RobotModel′}	A set of simplified robot models. In a simplified model, the particle distribution is converted into a Gaussian polar vector (see **??**).

5.2 Particle Filter

This section describes the architecture of the particle filter implementation, common to both the simulation implementation and the C firmware. Table 5.2 lists the data types present. When a Zebro i receives a beacon from Zebro j, it uses the information in the beacon (see table 5.3c) to update its RobotModels (see table 5.3b).

5.2.1 Motion Model

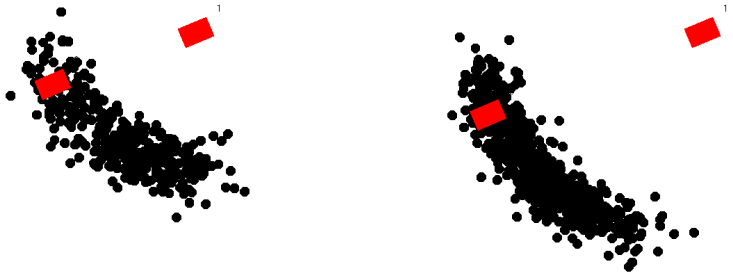
First, we apply the motion model. This requires a time delta since the last update, and a displacement vector. The time delta is calculated from the internal clock. The relative displacement vector $\Delta x'_{ij}$ is calculated by subtracting the displacements of i from those of j, since j's last beacon.

$$\Delta x'_{ij} = \Delta x_{ij} - \Delta x_{jj} \tag{5.1}$$

This vector gives a best estimate of the two Zebro's relative displacement. We then approximate the error on the distance estimation. Our current model for the motion takes the time delta, and the magnitude of the displacement vector, and scales them with the factors mentioned in table 5.1, and takes their Pythagorean sum to combine the two variances:

$$\sigma_x^2 = \sqrt{(|\Delta x'_{ij}| * \texttt{distanceVar})^2 + (\Delta t * \texttt{timeVar})} \tag{5.2}$$

Our final motion model is a Gaussian vector, with mean $\Delta x'$, and variance σ_x^2. In order to apply the motion model, we iterate over every particle, sample the motion model, and add the resulting vector to the particles'



(a) (b)

Figure 5.1: The motion model moves the particle cloud along along with the relative displacement vector between the two Zebros. After applying the motion model, the particles are resampled using the probability distribution generated from the distance measurement. The particles show the right Zebro's probability distribution for the position of the left Zebro.

position. See figure 5.1 for a demonstration of what this looks like.

5.2.2 Weight Updates

In order to resample using SIR (see section 4.1.7) due to some event E we need a weighting function $w_E : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$. This weighting function assigns a weight to every point in space, with the weight being proportional to the chance that the tracked object is at that position, given the occurrence of E. We choose to use the term weight instead of probability because weights do not necessarily need to sum to one and therefore do not need to be normalized.

$w_E(\vec{x})$ is directly derived from the characterization of the distance measurements. At the moment of writing, we do not yet have a characterization of the ranging hardware, and will therefore assume it to be Gaussian. This means that in our simulation, we use

$$w_E(\vec{x} \mid d, \sigma^2) = \exp\left[\frac{(|\vec{x}| - d)^2}{2\sigma^2}\right] \tag{5.3}$$

where d is the distance measurement, and σ is some predetermined variance. This is essentially a Gaussian distribution without the normalization factor. See algorithm 1 for the resampling algorithm pseudocode.

5.3 Cooperative Localization

Once we have a correct implementation for the particle filter, it is easy to extend this to cooperative localization. When performing cooperative localization, we allow the robots to share information about their own internal models, and use that information to improve the models of others. As long as the information they share can be converted into a weighting function, we can resample against this weighting function.

Algorithm 1 SIR Resampling. The argument w_E is a weighting function and p is a set of particles with each particle being a 2D vector.

procedure RESAMPLE(w_E, p)
  initialize an empty vector v of the same size as p ▷ v will store the weights
  $s \leftarrow 0$ ▷ s will hold the cumulative sum of the weights
  for every particle $\vec{p}_i$ **do**
    $v_i \leftarrow s$
    $s \leftarrow s + w_E(\vec{p}_i)$
  end for
  initialize an empty set of particles p'
  for every new particle $\vec{p}'_i$ **do**
    $q \leftarrow \text{Uniform}(0, s)$ ▷ Randomly pick a value from the CDF
    $k \leftarrow$ the largest index of v such that $v_k < q$
    $\vec{p}'_i \leftarrow \vec{p}_k$
  end for
end procedure

We are currently still experimenting with different methods of cooperative localization. One approach we developed that shows promising results is to share a Gaussian approximation of all RobotModels (table 5.3b). We found that this allows robots to triangulate positions, thereby improving accuracy. This is especially effective in large swarms.

One issue we found is that this could create feedback loops of false data. Our solution is to only use second-hand data if the variance on the second-hand data is smaller than that of the first-hand data:

$$\sigma_{ik} > \sqrt{\sigma_{ij}^2 + \sigma_{jk}^2} \tag{5.4}$$

where σ_{ab} is the standard deviation on the Zebro a's position estimate for Zebro b.

Whether or not this approach will be viable on the final hardware remains to be seen, as we do not know anything yet about e.g. the available bandwidth.

5.4 Compass

As explained in section 4.2.1, our initial efforts were focused on the HMC5983 with a breakout board. Communication with the chip is done via I2C, which we tested with an Arduino. After calibrating the magnetometer, we examined the sensor data. The sensor data is given in 3 axes; x, y and z. As we are only interested in the changes of orientation in the horizontal ('parallel to the surface'), we will preform the arctangent (`atan2` in arduino code) of the two relevant axis. The results were at first sight correct but, unfortunately, some adverse effects were found which will be discussed below.

5.4.1 Inaccuracies

The first problem has to do with the characteristics of digital magnetometers. The compass indicates horizontal and vertical (yaw and pitch) deviations in all three axis, serving as 3D compass. This causes unwanted results, because we need the horizontal orientation (the yaw, with respect to the north) and our sensor output is also dependent on a vertical displacement (pitch). Although this is logical, as vertically rotating the sensor 180 degrees (changing the pitch 180 degrees) should give the same result as horizontally (changing the yaw 180 degrees), we need an independence of vertical rotations. Because if we were to implement this sensor, huge inaccuracies (around 90 degrees) could occur if the Zebros moved on different slopes.

The second problem is that the magnetic field is disturbed indoors. When rotating the sensor, we found that it might detect a 180 degree rotation as anywhere between 150 and 210 degrees. Although the measurements outside were accurate, the module should also be able to work indoor, according to **[S-4]**. This problem is solved in smartphones via a concept called sensor fusion [46]. Sensor fusion combines the sensor data of multiple sensors such that the resulting information is more accurate.

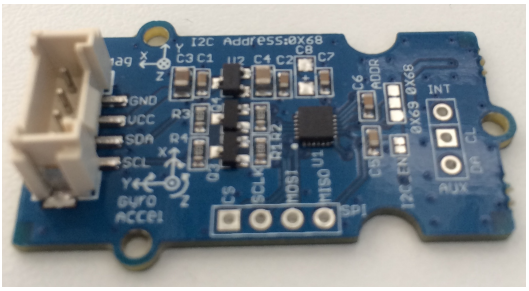
5.4.2 9DOF Sensors

As mentioned above, sensor fusion allows for more information concerning the heading or bearing of the Zebro. The heading or bearing are in this thesis used as the same variable; the direction of the Zebro with respect to the magnetic north. In most smartphone's, 9DOF sensors are used to accurately measure the orientation. DOF (Degrees Of Freedom) stand for the physically independent parameters that define its configuration[47]. 9DOF sensors can be used as an *inertial measurement unit* (IMU), a unit which can measure the yaw, pitch and roll of its body. The 9DOF sensors consist of:

- triple axis magnetometer
- triple axis gyroscope
- triple axis accelerometer

Adding these up, we get the 9 degrees of freedom. There also exists a 10DOF, equipped with a barometric pressure/altitude sensor but this is abundant for our purposes. Recall that we are only interested in the yaw of the Zebro. Sensor fusion is used to combine the magnetometer, gyroscope and accelerometer to accurately determine the horizontal projection of the magnetic field vector. Changing the tilt of the Zebro will no longer be dependent on the output, as we can see this change also affect the gyroscope and accelerometer.

Figure 5.2: Grove IMU 9DOF V2.0



Hardware and software

The 9DOF sensor we will use is the MPU9250, which is relatively cheap and widely used. The breakout circuit is shown in figure 5.2. I2C is used as communication protocol and an Arduino is used to test the sensor. The code used for sensor fusion is provided the open source RTIMULib[48]. Sensor fusion is implemented to accurately determine the yaw (orientation with respect to the magnetic north).

Using this library, we were able to get accurate compass readings, but only after manually calibrating the sensor. Calibration entails rotating the sensor along three axes, such that its maximum values can be saved to an EEPROM file. The only disadvantage is that each sensor has an unique calibration, which is time consuming for the production of Zebros on a large scale. This could be automated to save time and money.

6 Results and Conclusion

Our efforts are focused on two pieces of software; the simulation and the final firmware for the Zebro. We will discuss the performance of these pieces of software individually, draw conclusions, and make recommendations for for future work.

One common theme in our discussion is that it is difficult for us to make any guarantees about the performance of the final module in the absence of significant empirical data from the other submodules. The particle filter seems to be sensitive to parameters that are both outside our control and unknown to us at the time of writing.

6.1 Localization

Using our simulation, we were able to confirm that a particle filter is a promising method of robot localization, and it is especially suited for the DeciZebro. Unfortunately, it is not a silver bullet, as we found its performance in the simulation to be highly dependent on the configuration of the simulation. This makes it difficult for us to draw any meaningful conclusions about the projected performance of our module in the absence of empirical data from the other submodules.

The fact that convergence seems to be dependent on a combination of factors, combined with the large number of parameters we can change (see table 5.1), make it difficult to determine under what conditions the particle filter converges. As shown in figure 6.1, sometimes localization fails under a combination of parameters that individually do not impact convergence. Cooperative localization improved convergence in most cases, but would occasionally lead to worse results. The only reliable way we found to improve convergence was increasing the number of particles. In fact, with a sufficient number of particles, virtually all configurations converge.

This again highlights the difficulty of working without empirical data. In the simulation, for example, we generally used a beacon frequency of 1 Hz, but the final frequency might be an order of magnitude off. Once we are able to test our code on the firmware we will be able to see what number of particles we can use for filtering, which is instrumental to tweaking the other parameters. As we get more concrete results from the implementation of the other modules in the future, we will be able to focus our efforts on investigating configurations that mimic those of the final implementation.

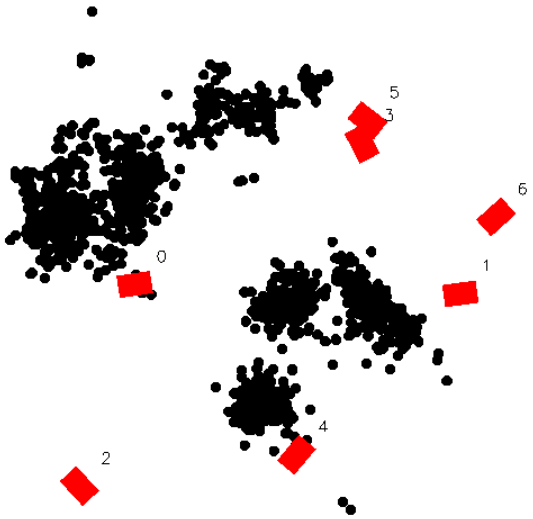
6.2 Firmware

Aside from the simulation, we finished a particle filter implementation for the Zebro firmware. Its code can be found on the Zebro git repository. Unfortunately, we have not yet been able to test the firmware's performance on the final hardware, as the other submodules were not yet functional at the deadline of this thesis. We could run the code on a regular computer, but this would likely not yield accurate performance measurements for the following reasons:

- Running the code with mock beacons for testing purposes means that all input and output data is known at compile time. This allows the compiler to perform many optimizations that would not be valid for the final code. While we did not extensively test whether this was the case, early performance measurements were so good that they might have been unreliable. Running the code without optimizations would also not provide accurate performance estimates, as we will definitely enable optimizations when compiling the final firmware.

- The compiler used to produce AVR code (AVR GCC) is different from the compiler that we use to produce x86 code for the computer (Clang). Different compilers can have different performance characteristics and perform different optimizations.

(a) A swarm successfully arranging itself in a crystal lattice structure



(b) Zebros failing to adequately localize other units

Figure 6.1: Examples of successful and unsuccessful localization following from the same initial positions. The black dots represent unit 6's knowledge about its surroundings. In the second figure, the number of particles was decreased from 1000 to 400 and the constant probability was increased from 0 to 0.1. Individually, neither of these changes is problematic. Only when these two changes are combined does localization fail.

6.3 Conclusion

All in all, we developed a promising approach for localization on the Zebro. Based on the current results, we believe that a particle filter was the right choice, and that it will allow us to perform accurate localization on the DeciZebro. The fact that we wrote a simulation also showed potential problems with the particle filter: there are many parameters to tweak, not to mention different cooperative localization schemes, which will significantly influence the converging. Although this problem is solved by simply increasing the number of particles, adjusting the particle filter is crucial for optimization purposes. Unfortunately, this adjusting can't be done yet as we rely on other submodules which have not yet been finished. This flexibility was one of the reasons of choosing this particle filter and has proven to function properly in the simulation. In the simulation, the particle filter has solved the localization problem within the given requirements and derived criteria. For this to be able to function in reality, tweaking parameters is essential and is likely to also work due to the high flexibility of the particle filter.

Future Work

Once the hardware is known, we might find that the problem is especially suited to using analytical methods or Kalman filtering. The primary reason we chose particle filtering is that it is the most general solution, but under the right circumstances the other methods could provide the same results at much less computational cost. This is an interesting avenue for future research efforts.

We will continue to experiment with cooperative localization schemes. Once we know parameters like available bandwidth we can focus our efforts on finding an efficient way to perform cooperative localization. The fact that cooperative localization is so implementation-dependent makes it difficult to find a general solution, but future research in the specific case of the Zebro might yield new approaches to localization altogether.

We also found that using a compass was harder than anticipated. It seems like a compass would provide valuable information to other Zebro modules, like the mapping module currently in development. For that reason, we believe that finding a way to accurately provide compass measurements to the entire Zebro, preferably without manual calibration, would aid the development of future Zebro modules.

Bibliography

[1] Zebro Team. (2017). Zebro project, [Online]. Available: https://www.tudelft.nl/d-dream/teams/zebro-project/.

[2] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Transactions on Automatic Control*, vol. 51, no. 3, pp. 401–420, Mar. 2006, ISSN: 0018-9286. DOI: 10.1109/TAC.2005.864190.

[3] X. S. Zhou and S. I. Roumeliotis, "Robot-to-robot relative pose estimation from range measurements," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1379–1393, Dec. 2008, ISSN: 1552-3098. DOI: 10.1109/TRO.2008.2006251.

[4] J. Strader, Y. Gu, J. N. Gross, M. D. Petrillo, and J. Hardy, "Cooperative relative localization for moving uavs with single link range measurements," in *2016 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, Apr. 2016, pp. 336–343. DOI: 10.1109/PLANS.2016.7479718.

[5] A. Cornejo and R. Nagpal, "Distributed range-based relative localization of robot swarms," in *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, H. L. Akin, N. M. Amato, V. Isler, and A. F. van der Stappen, Eds. Cham: Springer International Publishing, 2015, pp. 91–107, ISBN: 978-3-319-16595-0.

[6] M. Coppola, K. McGuire, K. Y. W. Scheper, and G. C. H. E. de Croon, "On-board bluetooth-based relative localization for collision avoidance in micro air vehicle swarms," *CoRR*, vol. abs/1609.08811, 2016. [Online]. Available: http://arxiv.org/abs/1609.08811.

[7] P. Ekberg, "Swarm-intelligent localization," Master's thesis, Uppsala Universitet, 2009.

[8] Z. Chen, "Bayesian filtering: from kalman filters to particle filters, and beyond," McMaster University, Tech. Rep., 2003. [Online]. Available: http://soma.crl.mcmaster.ca/%5C~%7B%7Dzhechen/ieee%5C_bayes.ps.

[9] E. Wan, "Sigma-point filters: An overview with applications to integrated navigation and vision assisted control," in *2006 IEEE Nonlinear Statistical Signal Processing Workshop*, Sep. 2006, pp. 201–202. DOI: 10.1109/NSSPW.2006.4378854.

[10] D. Simon, *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley-Interscience, 2006, ISBN: 0471708585.

[11] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, no. 1-2, pp. 99–141, 2000.

[12] D. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello, "Bayesian filtering for location estimation," *IEEE Pervasive Computing*, vol. 2, no. 3, pp. 24–33, Jul. 2003, ISSN: 1536-1268. DOI: 10.1109/MPRV.2003.1228524. [Online]. Available: http://dx.doi.org/10.1109/MPRV.2003.1228524.

[13] F. Gustafsson, "Particle filter theory and practice with positioning applications," *IEEE Aerospace and Electronic Systems Magazine*, vol. 25, no. 7, pp. 53–82, Jul. 2010, ISSN: 0885-8985. DOI: 10.1109/MAES.2010.5546308.

[14] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, Feb. 2002, ISSN: 1053-587X. DOI: 10.1109/78.978374.

[15] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, "The cricket location-support system," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '00, Boston, Massachusetts, USA: ACM, 2000, pp. 32–43, ISBN: 1-58113-197-6. DOI: 10.1145/345910.345917. [Online]. Available: http://doi.acm.org/10.1145/345910.345917.

[16] Rubenstein and Nagpal, *Kilobot: A robotic module for demonstrating behaviors in a large scale collective*, 2010. [Online]. Available: https://dash.harvard.edu/handle/1/5504015.

[17] K. Kouwenhoven and S. Verkamman, *Development of omnidirectional distance sensing module for the decizebro*, Technische Universiteit Delft, 2017.

[18] M. Wolleswinkel and R. Schotman, *Localization and communication module for decizebros*, Technische Universiteit Delft, 2017.

[19] *Decizebro module interface specifications*, The Zebro Project, 2017.

[20] P. de Vaere and D. Booms, *Zebrobus standard specification*, v1, The Zebro Project, Jun. 2016.

[21] F. Forno, G. Malnati, and G. Portelli, "Design and implementation of a bluetooth ad hoc network for indoor positioning," *IEE Proceedings - Software*, vol. 152, no. 5, pp. 223–228, Oct. 2005, ISSN: 1462-5970. DOI: 10.1049/ip-sen:20045027.

[22] D. Scheerens, *Practical indoor localization using bluetooth*, Jan. 2012. [Online]. Available: http://essay.utwente.nl/61496/.

[23] E. Dahlgren and H. Mahmood, *Evaluation of indoor positioning based on bluetooth smart technology*, Jun. 2014. [Online]. Available: https://goo.gl/KYPhUO.

[24] B.-C. Min, E. T. Matson, and J.-W. Jung, "Active antenna tracking system with directional antennas for enhancing wireless communication capabilities of a networked robotic system," *Journal of Field Robotics*, vol. 33, no. 3, pp. 391–406, 2016, ISSN: 1556-4967. DOI: 10.1002/rob.21602. [Online]. Available: http://dx.doi.org/10.1002/rob.21602.

[25] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan, "Beepbeep: A high accuracy acoustic ranging system using cots mobile devices," in *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, ser. SenSys '07, Sydney, Australia: ACM, 2007, pp. 1–14, ISBN: 978-1-59593-763-6. DOI: 10.1145/1322263.1322265. [Online]. Available: http://doi.acm.org/10.1145/1322263.1322265.

[26] J. Pugh, X. Raemy, C. Favre, R. Falconi, and A. Martinoli, "A fast onboard relative positioning module for multirobot systems," *IEEE/ASME Transactions on Mechatronics*, vol. 14, no. 2, pp. 151–162, Apr. 2009, ISSN: 1083-4435.

[27] H. Balakrishnan, R. Baliga, D. Curtis, M. Goraczko, A. Miu, B. Priyantha, A. Smith, K. Steele, and S. T. K. Wang, *Lessons from developing and deploying the cricket indoor location system*, Nov. 2003. [Online]. Available: http://cricket.csail.mit.edu/V1Exp.pdf.

[28] A. Smith, H. Balakrishnan, M. Goraczko, and N. Priyantha, *Tracking moving devices with the cricket location system*, Jun. 2004. [Online]. Available: http://nms.csail.mit.edu/papers/tracking%5C_mobisys04.pdf.

[29] C. M. Yu, S. K. Hung, and Y. C. Chen, "Forming mesh topology for bluetooth ad hoc networks," in *2013 IEEE International Symposium on Consumer Electronics (ISCE)*, Jun. 2013, pp. 123–124. DOI: 10.1109/ISCE.2013.6570141.

[30] J. A. Prasetyo, A. Yushev, and A. Sikora, "Investigations on the performance of bluetooth enabled mesh networking," in *2016 3rd International Symposium on Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS)*, Sep. 2016, pp. 56–61. DOI: 10.1109/IDAACS-SWS.2016.7805786.

[31] A. Jedda, A. Casteigts, G.-V. Jourdan, and H. T. Mouftah, "Bluetooth scatternet formation from a time-efficiency perspective," *Wireless Networks*, vol. 20, no. 5, pp. 1133–1156, 2014, ISSN: 1572-8196. DOI: 10.1007/s11276-013-0664-z. [Online]. Available: http://dx.doi.org/10.1007/s11276-013-0664-z.

[32] Y. Wei, M. Song, and J. Song, "An aodv-improved routing based on power control in wifi mesh networks," in *2008 Canadian Conference on Electrical and Computer Engineering*, May 2008, pp. 001 349–001 352. DOI: 10.1109/CCECE.2008.4564760.

[33] S. Farahani, Ed., *ZigBee Wireless Networks and Transceivers*. Burlington: Newnes, 2008, ISBN: 978-0-7506-8393-7. [Online]. Available: http://www.sciencedirect.com/science/book/9780750683937.

[34] S. Choudhury, P. Kuchhal, and R. Singh, "Zigbee and bluetooth network based sensory data acquisition system," *Procedia Computer Science*, vol. 48, pp. 367–372, 2015, ISSN: 1877-0509. DOI: http://dx.doi.org/10.1016/j.procs.2015.04.195. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915007048.

[35] A. Fernandes, M. S. Couceiro, D. Portugal, J. Machado Santos, and R. P. Rocha, "Ad hoc communication in teams of mobile robots using zigbee technology," *Computer Applications in Engineering Education*, vol. 23, no. 5, pp. 733–745, 2015, ISSN: 1099-0542. DOI: 10.1002/cae.21646. [Online]. Available: http://dx.doi.org/10.1002/cae.21646.

[36] Z. Yi, H. Hou, Z. Dong, X. He, Z. Lv, C. Wang, and A. Tang, "Zigbee technology application in wireless communication mesh network of ice disaster," *Procedia Computer Science*, vol. 52, pp. 1206–1211, 2015, ISSN: 1877-0509. DOI: http://dx.doi.org/10.1016/j.procs.2015.05.159. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S187705091500959X.

[37] D. International. (2015). Wireless mesh networking, zigbee vs. digimesh, [Online]. Available: https://www.digi.com/pdf/wp%5C_zigbeevsdigimesh.pdf.

[38] F. Cai, E. Farantatos, R. Huang, A. P. S. Meliopoulos, and J. Papapolymerou, "Self-powered smart meter with synchronized data," in *2012 IEEE Radio and Wireless Symposium*, Jan. 2012, pp. 395–398. DOI: 10.1109/RWS.2012.6175381.

[39] R. E. Kalman, "A new approach to linear filtering and prediction problems," *ASME Journal of Basic Engineering*, 1960.

[40] L. Kleeman, "Understanding and applying kalman filtering," Department of Electrical and Computer Systems Engineering, Monash University, Clayton.

[41] E. A. Wan and R. V. D. Merwe, "The unscented kalman filter for nonlinear estimation," 2000, pp. 153–158.

[42] J. D. Hol, T. B. Schon, and F. Gustafsson, "On resampling algorithms for particle filters," in *2006 IEEE Nonlinear Statistical Signal Processing Workshop*, Sep. 2006, pp. 79–82. DOI: 10.1109/NSSPW.2006.4378824.

[43] A. Patki, "Particle filter based slam to map random environments using "irobot roomba"," Master's thesis, Vanderbilt University, 2011.

[44] A. Jarrah, M. M. Jamali, and S. S. S. Hosseini, "Optimized fpga based implementation of particle filter for tracking applications," in *NAECON 2014 - IEEE National Aerospace and Electronics Conference*, Jun. 2014, pp. 233–236. DOI: 10.1109/NAECON.2014.7045808.

[45] Honeywell, *3-axis digital compass ic*.

[46] U. Shala and A. Rodriguez, *Indoor positioning using sensor-fusion in android devices*, 2011.

[47] L. Danisch, K. Englehart, and A. Trivett, "Spatially continuous six degree of freedom position and orientation sensor," *Sensor Review*, vol. 19, no. 2, pp. 106–112, 1999.

[48] Richards-tech. (2017). Rtimulib, [Online]. Available: https://github.com/RPi-Distro/RTIMULib.