Karlsruhe Institute of Technology (KIT)
Department of Computer Science
Institute for Anthropomatics and Robotics
Chair for Intelligent Sensor-Actuator-Systems (ISAS)
Prof. Dr.-Ing. Uwe D. Hanebeck

Practical course

Laboratory Research Project: Hands-on Anthropomatics

July 28, 2017

Lukas Bernreiter, Stefan Fehrenbach, Michael Kopf

Simultaneous Localization and Mapping using Occupancy Grid-based Particle Filters and Ultrasonic sensors

ISAS.uni-karlsruhe.de

Simultaneous Localization and Mapping using Occupancy Grid-based Particle Filters and Ultrasonic sensors

– Practical course: Laboratory Research Project: Hands-on Anthropomatics–

Lukas Bernreiter, Stefan Fehrenbach, Michael Kopf

July 28, 2017

Abstract

The ability for a robot to localize itself is the fundamental concept for autonomous robotic applications. The problem complexity rises quite high for localization in unknown areas. This comes from the fact that a robot needs a map of the environment in order to localize itself. However, in order to create a map a robot needs to localize itself. This problem statement is known as the Simultaneous Localization And Mapping (SLAM) problem.

Furthermore, this work deals with the SLAM problem by using so called Particle Filters (PFs) and ultrasound sensors. PFs use particles, which are samples, to estimate the current pose, i.e. the position and orientation of the robot.

There are two essential parts that need to be implemented, namely the system and the measurement model. This work mainly deals with the measurement model, i.e. the relationship between a state and a measurement. Nevertheless, it still enhances the system model by incorporating an inertial measurement unit.

Additionally, this work also emphasizes on the problem of a fully autonomous exploration of an unknown environment. Here the robot needs to perform a dynamic path planning algorithm and if necessary to adapt its trajectory.

Finally, this work is evaluated using a mobile robot and a camera. The camera is used to track the robot's real position which is then compared to the estimated trajectory.

Simultane Lokalisierung und Kartographierung mit Grid-basierten Partikelfilter und Ultraschall-Sensoren

– Praktikum: Forschungsprojekt: Anthropomatik praktisch erfahren –

Lukas Bernreiter, Stefan Fehrenbach, Michael Kopf

July 28, 2017

Zusammenfassung

Die Fähigkeit eines Roboters sich in einer Umgebung zu lokalisieren ist eine der fundamentalen Funktionalitäten für autonome Anwendungen. Dabei steigt die Problemkomplexität sehr stark an, wenn dieses für beliebige und unbekannte Umgebungen funktionieren soll.

Das resultiert daraus, dass ein Roboter eine Karte der Umgebung benötigt, um sich selbst lokalisieren zu können. Allerdings wird die aktuelle Position des Roboters benötigt, um eine Karte aufzubauen. Dieses Problem wird im Allgemeinen als das simultane Lokalisierungs- und Kartographierungs- (SLAM-) Problem bezeichnet.

Diese Arbeit beschäftigt sich mit der Lösung des SLAM-Problems mithilfe von sogenannten Partikel-Filtern (PFs). PFs verwenden Partikel, sprich Stichproben, um die aktuelle Pose, welche aus Position und Orientierung des Roboters besteht, zu schätzen.

Für die Verwendung sind zwei grundsätzliche Bestandteile zu implementieren, nämlich das System- und das Messmodell. Dabei liegt der Fokus dieser Arbeit hauptsächlich auf den Messmodellen, welche den Zusammenhang von Zustand und Messung beschreiben.

Zusätzlich wird in dieser Arbeit noch das Problem der autonomen Erkundung erarbeitet. Solche Algorithmen müssen eine dynamische Pfadplanung durchführen und sich gegebenenfalls auch dynamisch adaptieren.

Zum Schluss wird diese Arbeit noch mittels eines mobilen Roboters und einer Deckenkamera evaluiert. Diese Kamera wird verwendet, um die aktuelle tatsächliche Position des Roboters zu errechnen, welche dann später mit der geschätzten Route verglichen wird.

Contents

Acronyms

SLAM *Simultaneous Localization and Mapping*

KF *Kalman Filter*

EKF *Extended Kalman Filter*

UKF *Unscented Kalman Fitler*

IMU *Inertial Measurement Unit*

PF *Particle Filter*

PDF *Probability Density Function*

MCL *Monte Carlo Localization*

ISAS *Intelligent Sensor-Actuator Systems*

GUI *Graphical User Interface*

NLE *Nonlinear Estimation Toolbox*

MC *Monte Carlo*

DMP *Digital Motion Processor*

MEMS *Micro-Electro-Mechanical-System*

RPY *Roll Pitch Yaw*

MSE *Mean Squared Error*

PID *Proportional-Integral-Derivative*

Notation

Sets

$\mathbb{R}$	The set of real numbers.
$\mathbb{N}$	The set of natural numbers.

Indices

$(.)^{[m]}$	Quantity of particle m.
$(.)_k$	Quantity at time step k.

Scalars, Vectors, Matrices

x	A scalar.
$\underline{x}$	A vector in Euclidian space.
$\mathbf{A}$	A matrix.

Density Functions

$f(\underline{x})$	The probability density function of $\underline{x}$.
$f_k(\underline{x})$	The probability density function of $\underline{x}$ at time step k.

List of Figures

1 Introduction

Robotic localization and the process of mapping the environment is an heavily active research field. Localization as well as mapping problems are of great significance in both academia and industry. The field of robotic localization in general is not a new research topic. It is rather a field which has been studied for a long time and is already extensively covered by literature. A good overview of different kinds of localization methods and problems is for example given in [Thr+02; SV00].

1.1 Motivation

The ability of a robot to localize itself is one of the most important skills of autonomous mobile robots. This applies to a variety of different robots operating on land, sea or air as well as commercial, research and military robots. This holds especially true for robotic applications where it is extremely dangerous or incredibly hard for humans to operate, e.g. exploration of mines or terrain mapping in space.

Typical examples for commercial robots are vacuum cleaners and lawn mowers. However, as far as these commercial robots are concerned, they typically try to avoid or to minimize the localization problem to a certain degree.

When a robot operates in an unknown area, e.g. robots designed for search and rescue missions, somehow need to have the functionality to create a map of the environment and to determine its current position. In order to localize itself, a mobile robot needs on the one hand information about it surroundings. In other words it needs a map, which is obtained by sensors, such as range finders or cameras. On the other hand and this is the main problem with mapping, a robot needs to localize itself in order to build a map.

In general this problem is known as the *Simultaneous Localization and Mapping* (SLAM) problem.

The simplest form of localization is called dead reckoning or deduced reckoning. Dead reckoning does not need to have a map for the localization but rather this approach solely relies on odometry information and traces the relative movements from the initial starting point. In other words, dead reckoning does not include any measurements from the environment and thus, this approach suffers from an error which accumulates over time. Figure 1 illustrates this behavior by an error-prone estimated robot path.

Since the map is not known, it is not possible to use any static localization methods either as these techniques need to know the location of the landmarks in the environment.

Therefore, considering all the above stated requirements and problems, there is a need of an approach which deals with the localization problem in unknown environments. There are several methods which can be used to solve the SLAM problem. This work deals with the SLAM problem by using a so called grid-based *Particle Filter* (PF) as well as ultrasound sensors.

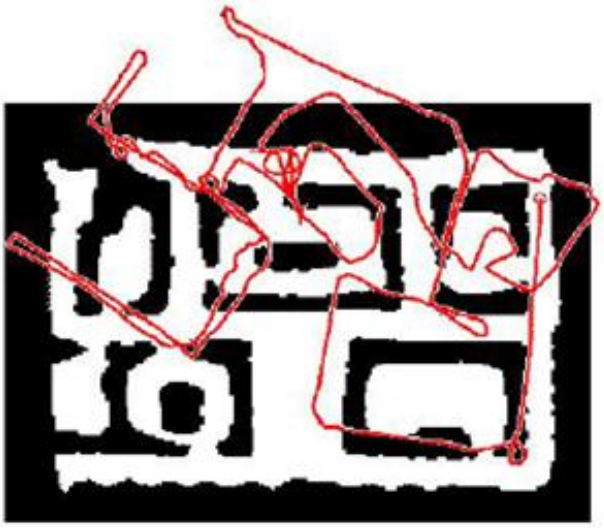


Figure 1: An estimated path of a mobile robot using dead reckoning [Thr+02].

1.2 Related Work

Traditionally, the classic approach to solve the SLAM problem is to use the *Kalman Filter* (KF) approach. The main problem with the vanilla KF is that it makes two main assumptions. First, that the used system and measurement models are linear models. Second, in order to be an optimal filter, the noise resulting from the sensors as well as from the system is Gaussian.

These assumptions are actually more like restrictions than assumptions. Besides, a typical mobile robot trajectory is in most cases not even close to a linear trajectory. This is the reason why different variations of the KF, such as the *Extended Kalman Filter* (EKF) and the *Unscented Kalman Fitler* (UKF) exist. Typically, these variations perform linearization techniques, such as Taylor series expansions which, however, introduces a linearization error to the already existing errors.

Therefore, this and some other reasons motivated research to follow different approaches and to develop new variations to existing methods. One of these approaches is a sampling-based method and is known as *Monte Carlo Localization* (MCL). MCL is synomym for a PF, which this work uses for localization [Del+99]. PF will be the main focus of this work.

1.3 Previous Work

This project is a continuation of projects from previous semester terms. Therefore, it was neither necessary to design and build a mobile robot from scratch nor to deal with the communication with the robot. There already existed a mobile robot which was able to move omnidirectional in the environment. The robot included a set of ultrasound sensors and an *Inertial Measurement Unit* (IMU). For more information about the robot see chapter 3.

In addition, there was a software system which enabled to use Matlab to send control inputs over Wi-Fi to the robot. This software system was also used by the robot to publish the evaluated ultrasound data. However, the IMU was not usable at this moment.

Furthermore, the previous group already implemented a simulation software for the mobile robot in Matlab which partially includes a grid-based PF algorithm. In this implementation some of the models were not fully developed and approximated.

1.4 Outline

This work consists of the following parts:

Fundamentals Chapter 2 explains the SLAM problem in more detail. It gives a general introduction to Bayesian filtering and then emphasizes on the PF approach.

Hardware and Design Chapter 3 explains the used mobile robot. Additionally, it shows the implemented sensors for extracting odometry information of the robot as well as measurements of the environment. Furthermore, it deals with the system under consideration.

Project Scope Chapter 4 gives details about the project's structure and plan. It outlines the tasks as well as the milestones.

Implementation In chapter 5 the implementation and the used methods are explained. In more detail, it gives a good explanation of the models of the implementation. In addition, it briefly gives an overview of the used libraries and frameworks.

Results The chapter 6 provides results and comparisons between different approaches.

Conclusion Finally, chapter 7 discusses the project. Additionally, it gives an outlook for further projects.

2 Fundamentals

This chapter introduces the theoretical background of this work in more detail. Chapter 1 already introduced and motivated the SLAM problem. Section 2.1 defines the SLAM problem formally and thereby builds the theoretical background for this work. Next, a general introduction to Bayesian filtering is given in section 2.2. Finally, the rest of this chapter emphasizes on PFs and additionally utilized methods as well as how to apply these in order to solve the SLAM.

2.1 Simultaneous Localization and Mapping

A robot's pose or also called state $\underline{x}$ comprises the location as well as the orientation of the robot

$$\underline{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}.$$

These parameters can be used to determine a robot's location. Given a trajectory the location of the robot can be represented by a sequence of robot locations at discrete time points

$$\mathbf{X}_t = \{\underline{x_1}, \underline{x_2}, ..., \underline{x_t}\},$$

where $t \in \mathbb{N}$.

Additionally, a mobile robot typically needs some sort of control inputs which are used to send motion commands to the robot. These inputs can come from a teleoperator (i.e. human) or from an autonomous exploration algorithm. In most cases this is modelled and represented by the so called unicycle model as follows

$$\underline{u} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix},$$

where $\dot{x}$ and $\dot{y}$ represents the velocity in x- and y-direction, respectively. The $\dot{\theta}$ denotes the angular velocity of the mobile robot. Considering a trajectory of a mobile robot again, the form of the trajectory correlates to a set of input arguments at discrete points in time

$$\mathbf{U}_t = \{\underline{u_1}, \underline{u_2}, ..., \underline{u_t}\}.$$

Finally, a mobile robot takes measurements of the environment and its state by using internal as well as external sensors. Since no sensor is perfect and always contains some sort of noise,

a mobile robot contains a variety of sensors in most cases in order to compensate for sensor inaccuracies. Therefore, relying on one single sensor is in general not a good idea.

Typically, sensors can either be passive (measurement of the given environment) or active (sending signals to measure the environment). Commonly used passive sensors are depth- and RGB-cameras. Typically used active sensors are ultrasound sensors or laser scanners. The form of the measurements depends on the sensor that is being used, e.g. an ultrasound range finder sensor gives distance information. When a robot drives along a trajectory it collects a set of measurements of the environment or of itself

$$\mathbf{Z}_t = \{z_1, z_2, ..., z_t\}.$$

All this above mentioned components can be used to formally define the SLAM problem. Literature distinguishes between two types of the SLAM problem, namely the *full SLAM problem* and the *online SLAM problem* [SK16].

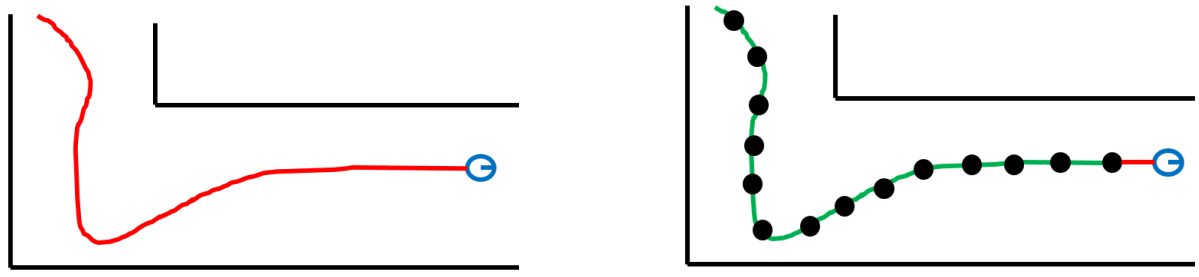
The full SLAM problem is concerned about estimating the posterior over the whole path

$$p(X_t, m|Z_t, U_t).$$

Whereas the online SLAM problem tries to estimate the current position of the mobile robot

$$p(x_t, m|Z_t, U_t).$$

The figure 2 illustrates the difference between these two problem statements. As a matter of fact this work only emphasizes on the online SLAM problem.



(a) (b)

Figure 2: The different kinds of the SLAM problem. a) The full SLAM problem estimates the entire path of the mobile robot (red). b) The online SLAM problem tries to estimate the current position and can then recursively create a trajectory (red) to the last known position (green) to discrete time points (black dots).

How to estimate this *Probability Density Function* (PDF) $p(x_t, m|Z_t, U_t)$ is the topic of the following section.

2.2 Bayesian Filtering

Having the SLAM problem formally defined, it is now possible to introduce a famous family of filters, the so called Bayesian filters. Filters such as the PF as well as the KF belong to this family.

Most localization algorithms are probabilistic, due to the fact that deterministic models would be in most cases far too complex to model and hence too computationally expensive.

Furthermore, many probabilistic approaches have a single underlying principle, namely the *Bayes' Theorem*

$$p(x|z,y) = \frac{p(y|x,z) \cdot p(x|z)}{p(y|z)}. \tag{1}$$

The key idea is how to estimate the robot's current position and orientation, i.e. the robot's state. For this, the term $bel(\underline{x}_t)$ is introduced which refers to the robots belief of its current location at time step t. This results from the fact that it is not possible to directly measure a robot's state but rather to infer it from measurements.

Generally speaking, a Bayesian filter implements the algorithm as shown in algorithm 1. On a closer look one can see the familiar form of Bayes' Theorem (cf. equation 1). [TBF05]

Algorithm 1 Bayes Filter

1: **procedure** BAYES_STEP($bel(x_{t-1}, u_t, z_t)$)
2: **for all** x_t **do**
3: $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) * bel(x_{t-1})dx_{t-1}$ ▷ Prediction Step
4: $bel(x_t) = \eta * p(z_t|x_t) * \overline{bel}(x_t)$ ▷ Filter Step
5: **end for**
6: **return** $bel(x_t)$
7: **end procedure**

A Bayesian filter essentially implements two steps, namely the prediction/time update step (line 4) and the filter/measurement update step (line 5).

The equation for the prediction step is also known as the Chapman-Kolmogorov equation and can be quite difficult to solve for general, nonlinear cases. This motivates the use of nonlinear, suboptimal filters.

2.3 Particle Filter

The fact that the reality is nonlinear, non-Gaussian and consists of a continuous state space limits the use of many filters. In other words, the restrictions of the real world enforces the use of suboptimal nonlinear filters instead of optimal KFs for example.

According to [RAG04], these filters can be divided into four groups: (1) analytic approximations; (2) numerical approximations; (3) multiple model filters and (4) sampling approaches.

In this work, the focus lies on a specific sampling approach namely the PF. In general, the idea behind PFs is to have a set of particles, whereby each particle contains an estimate of the current real state. PFs are similar to the UKF approach, the main difference is, however, that the samples (sigma points) are not determined deterministically but rather randomly.

The fundamental approach behind PFs is the *Monte Carlo* (MC) integration which states that an integral

$$I = \int f(x)dx$$

can be approximated by the sample mean

$$\widetilde{I} = \frac{1}{N}\sum_{m=1}^{N} f(x^{[m]}).$$

The approximation will converge to I for a large numbers of samples [RAG04].

An advantage of the use of PFs is that a sample $x^{[m]}$ can easily be propagated through a nonlinear mapping $g(x^{[m]})$. A general implementation follows algorithm 2. [TBF05]

Algorithm 2 Particle Filter

1: **procedure** UPDATE_STEP(χ_{t-1}, u_t, z_t)
2: **for** $m = 1$ to M **do**
3: sample $x_t^{[m]} \sim p(x_t|u_t x_{t-1}^{[m]})$
4: $w_t^{[m]} = p(z_t|x_t^{[m]})$ ▷ Importance Factor
5: $\bar{\chi} = \bar{\chi} + \langle x_t^{[m]}, w_t^{[m]} \rangle$
6: **end for**
7: **for** $m = 1$ to M **do** ▷ Resampling
8: draw i with probability $\propto w_t^{[i]}$
9: add $x_t^{[i]}$ to χ_t
10: **end for**
11: **return** χ_t
12: **end procedure**

A sample $x_t^{[m]} \sim p(x_t|u_t x_{t-1}^{[m]})$ can be obtained by first generating a sample from the noise density $v_{t-1}^{[m]}$ and then calculating $x_t^m = f_{t-1}(x_{t-1}^{[m]}, v_{t-1}^{[m]})$.

The importance factor or weight is essentially the likelihood, which represents the relation between the state $x_t^{[m]}$ and a measurement z_t. Both the sample and the importance factor are stored in a temporary set $\bar{\chi}$.

Finally, the particles are drawn with replacement from the temporary set and are added to the result set χ_t. In reference to section 2.2, before the resampling takes place, the particles are distributed according to $\overline{bel}(x_t)$ and after it took place approximately according to $bel(x_t)$.

An important fact of PFs is that if the number of particles approaches to infinity the filter actually converges to the optimal solution. Since this is not possible in a practical sense, it is considered as a suboptimal filter.

2.4 Occupancy Grid Maps

An approach to represent a local map is to use so called occupancy grids. An occupancy grid map is generally speaking a probabilistic, two-dimensional raster. Each cell in the map can either be unknown, free or occupied (see figure 3). An unknown state means that there is no information about this area available, i.e. a mobile robot has never received a measurement in this area.

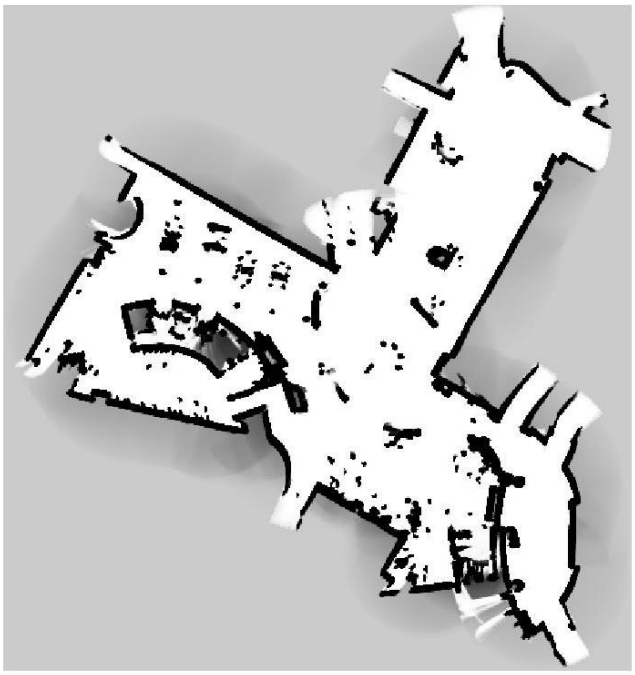


Figure 3: An example for an occupancy grid map. White areas are free cells, black black areas are detected obstacles and gray areas denote unknown areas [TBF05].

Every time when a new measurement is available, the occupancy grid map will be updated according to that measurement. More formally speaking, an occupancy grid m is defined by a set of cells:

$$m = \{m_{i,j}\},$$

where i, j is the index a grid cell. Each grid cell in the map is associated with a probability denoting its occupancy, e.g. 0 for occupied and 1 for free. Furthermore, practice showed that introducing unknown occupancy as a third state can be an advantage.

When a new measurement is available, it is necessary to compute the new probability:

$$p(m_{i,j}|z_{1:t}, x_{1:t}) = 1 - \frac{1}{1 + \exp\{l_{t,i,j}\}},$$

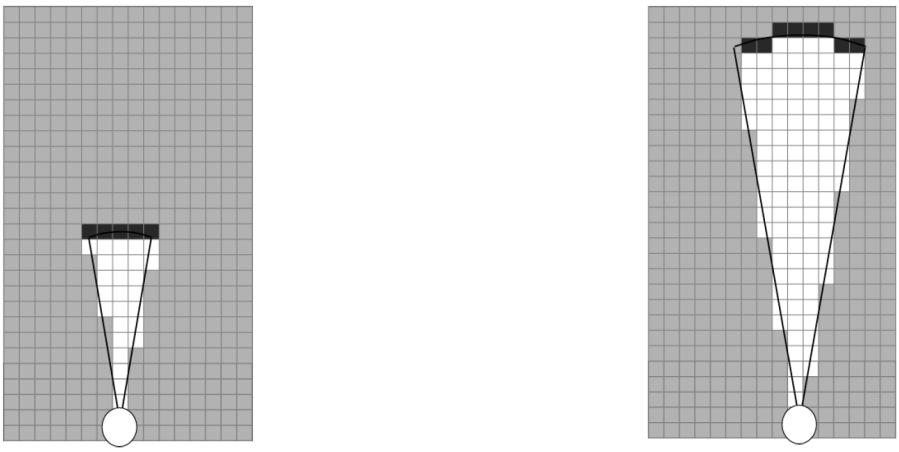
where

$$l_{t,i,j} = l_{t-1,i,j} + l(m_{i,j}|z_t, x_t) - l(m_{i,j}). \tag{2}$$

The equation 2 can be calculated recursively and is represented in the so called log odds form. The log odds form is computationally more efficient than using regular probabilities since it only needs to compute two sums rather than products.

The term $l(m_{i,j})$ is a constant factor and represents the prior of the grid map. Additionally, if this factor is set to zero then the term completely disappears [Thr+02].

Finally, the term $l(m_{i,j}|z_t, x_t)$ is also known as the inverse sensor model. The inverse measurement mainly depends on the used sensor. Since in this work ultrasound sensors were used, the inverse measurement model calculates the probability along a cone of a measurement. Given a concrete measurement z_t, this model identifies the cells in the occupancy grid map which are in the perceptual field of the sensor. figure 4 shows an example of the result of an inverse measurement calculation for an ultrasound sensor.



(a) (b)

Figure 4: Two examples of an inverse measurement [TBF05].

2.5 Grid-based Particle Filter

Grid-based PFs combine the idea of using occupancy grids and particles with individual states. Basically, each particle contains in addition to the robot's pose a local occupancy grid map (see figure 5). The underlying idea comes from the fact that the map is conditionally independent from the state. In other words, no matter where the robot is currently located this does not change the actual positions of any landmarks or obstacles.

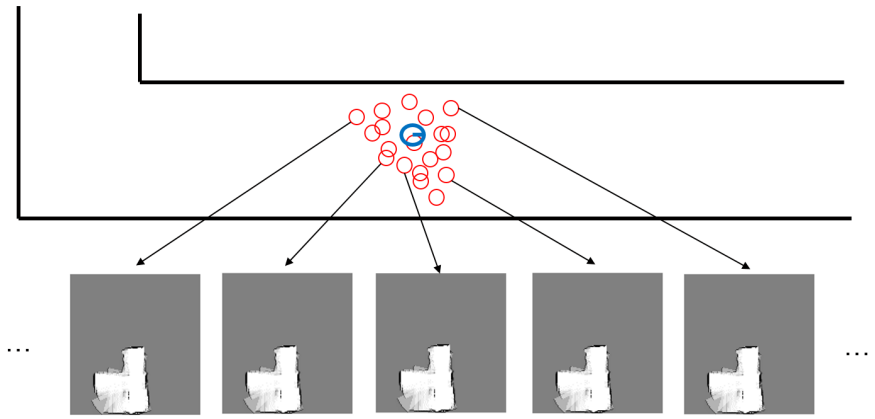


Figure 5: A robot (blue) with a set of particles (red) where each particle contains a reference to a local occupancy grid map (arrows).

This approach also works with maps other than occupancy grid maps and is known as *Rao-Blackwellization* or in a SLAM context as *FastSLAM* [Mon+02; TBF05].

3 Hardware and Design

The laboratory for *Intelligent Sensor-Actuator Systems* (ISAS) created a mobile walking robot for research purposes. The work presented in this documentation was implemented, tested and evaluated using this robot. This chapter gives more detailed insights about the robot used in this work and its functionality.

3.1 The ISAS Crawler Family

Many different versions of the crawler exist (see Figure 6). These robots are used for several research projects, such as stochastic control and SLAM. Furthermore, the chassis of these robots are completely 3D printed and can therefore be built quite fast.

(a) (b) (c)

Figure 6: Different kinds of crawlers. (a) (b) Older versions of the crawler. (c) Current version of the crawler.

One of the most recent versions of the crawler features four ultrasound sensors and one IMU. In addition, it features three different legs, wherein each can be controlled individually by four servo motors. This allows the robot to move straight, sideways and omnidirectional.

The main component of the version in this work is an Arduino Yùn. The Arduino Yùn features two different CPUs.

3.2 Odometry Information

The robot's odometry information is provided by an IMU of the *Micro-Electro-Mechanical-System* (MEMS) type. This IMU contains a *Digital Motion Processor* (DMP) which features eased access to the acceleration readings and calculation of the current orientation. Furthermore, the DMP already handles the noise of the gyroscopes and preprocesses the readings. In other words, it filters and corrects the values with e.g. the earth's rotation. The program used by the DMP needs to be flashed each time the Arduino boots. The DMP and the Arduino are communicating via I^2C (see figure 7).

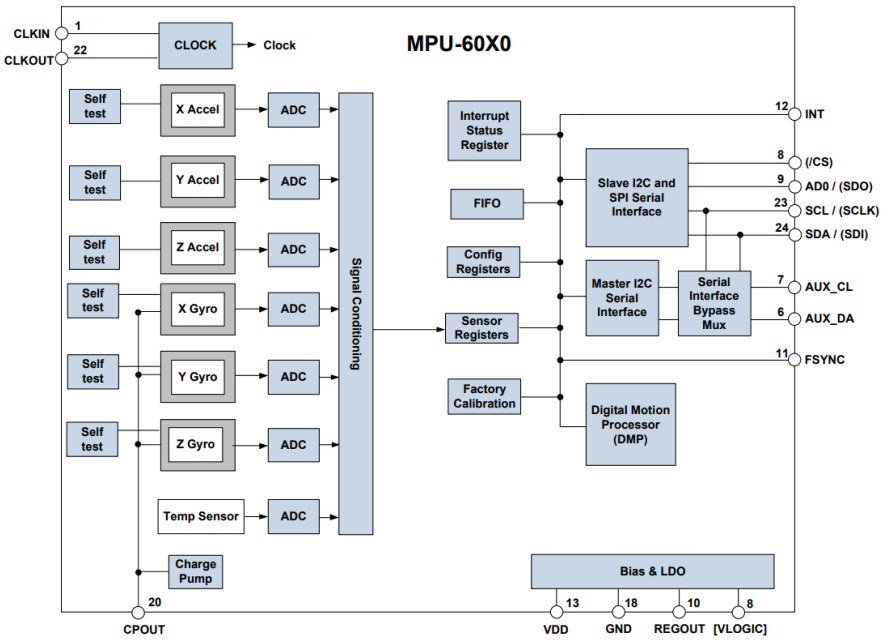


Figure 7: The MPU 6050 provides a three axis gyroscope and three axis accelerometer. The Arduino and the IMU are communicating via I^2C [Inv].

3.3 Environmental Measurements

The ISAS crawler used in this work utilizes four ultrasound sensors for measuring distances to nearby obstacles. Ultrasound sensors basically work by timing the difference between sending acoustic signals and receiving its reflected echo. This process is depicted in Figure 8.

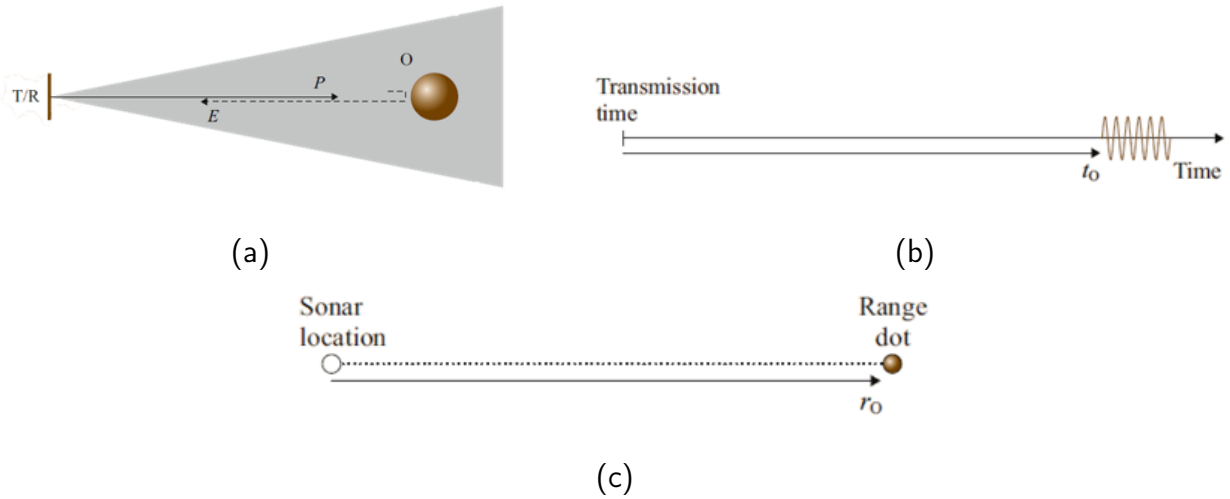


Figure 8: Principle of an ultrasound sensor. (a) An acoustic signal is sent towards an obstacle. (b) The measured time between sending an receiving an echo of the acoustic signal. (c) Distance calculation based on the measured time [SK16].

The problem with ultrasound sensors is that it is not possible to determine the exact location of an obstacle. It is merely possible to determine its distance on an arc.

3.4 System Design

In order to perform computations based on the introduced hardware, the hardware needs to be modeled. The two essential models for this are the system and the measurement model. The used models in this work are considered as dynamic, time-variant, time-discrete nonlinear models.

System model and space The state space of the system is considered to be a continuous state space, since $x \in \mathbb{R}$ and $y \in \mathbb{R}$. The system model can be separated into a linear and a nonlinear part. As far as the translations in x- as well as y-direction are considered, both are linear systems. The robot's orientation is the part of the system which makes it nonlinear.

The generative model of the system equation is given by

$$\underline{x}_{k+1} = a_k(\underline{x}_k, u_k, w_k),$$

where a_k is a nonlinear function, u_k a set of input parameters and w_k is assumed to be white, zero-mean Gaussian noise.

In general, the system model is used during the prediction step of the filter to represent the relation between two states during a time update.

Measurement model The measurement model is used to compute the likelihood in the filter step. Generally speaking, the likelihood represents the relation between the current state and a measurement.

At a time step k the robot receives a new measurement (see Figure 9) and uses a measurement model to calculate which measurement it would expect given the current state.

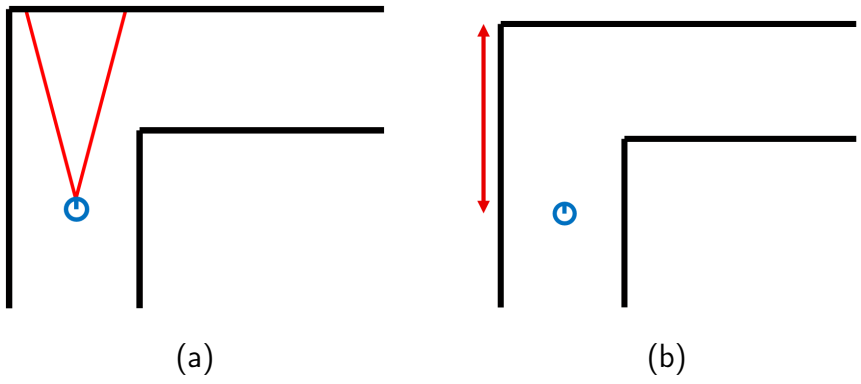


Figure 9: A robot (blue) performing an ultrasound measurement (red). (a) An ultrasound measurement represented by a cone. (b) Distance (red) calculation based on the measured time.

4 Project Scope

This chapter deals with the structure of the project. It covers the used tools for planning and managing this project.

The main part of project planning was done using Asana[1]. Asana is a freely available online platform for team collaboration and features the management of tasks and meetings.

4.1 Project Plan

The project was planned in total over 13 weeks and was divided into six sprints. Thereby, Asana provided a column-styled scrum board with the following stages. Table 1 illustrates the initial plan of the user stories over the course of the project. Each story was subdivided into one or more tasks and tracked in Asana.

Table 1: Project plan

			M1↓				M2↓					M3↓		
Stories	KW 18	KW 19	KW 20	KW 21	KW 22	KW 23	KW 24	KW 25	KW 26	KW 27	KW 28	KW 29	KW 30
Initial work													
Getting used to the code													
Simulation improvements								Intermediate presentation					Final presentation + Submission of the thesis
Performance improvements													
Implement measurement equations													
Evaluation and testing													
Integration of the IMU													
Autonomous Exploration													
Thesis													

Darkgreen: finished on time; Lightgreen: Started before schedule; Gray: Finished early; Orange: Delay/Too late

Furthermore, the project defined three milestones:

1. The existing simulation is well known, the necessary knowledge is acquired and the simulation results could be re-accomplished.
2. Simulation is substantially improved, measurement models are implemented and evaluated.
3. The IMU is integrated and the implementation of an autonomous exploration is finished.

[1] https://asana.com/

5 Implementation

In general, the software can be used to simulate a crawler and a world together as well as for controlling the real crawler.

This chapter covers the implementation of this work. An important part of the implementation is the *Nonlinear Estimation Toolbox* (NLE) toolbox [Ste]. The NLE toolbox provides the functions and methods to perform state estimation.

5.1 System Model

The kinematic model of the crawler can be used derive the system equation of the crawler [Wei08]. This results in following equation for the system model

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} cos(\theta_k) & -sin(\theta_k) & 0 \\ sin(\theta_k) & cos(\theta_k) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u_k^x \\ u_k^y \\ u_k^\theta \end{bmatrix} + \begin{bmatrix} w_k^x \\ w_k^y \\ w_k^\theta \end{bmatrix} .$$

Using this system model the crawler initially performs the translation and then the rotation. Furthermore, if $u_k^y = 0$ the crawler essentially implements a kinematic model of a differential drive robot [Wei08].

5.2 Measurement Model

An important implementation of this work is the measurement model for ultrasound sensors. The measurement model computes the expected measurement of a given estimate. Since this work uses PFs the measurement model computes an expected measurement for each particle.

This work introduces two measurement models for an ultrasound sensor, namely the so called *beam range finder model* and *likelihood filed range finder model*.

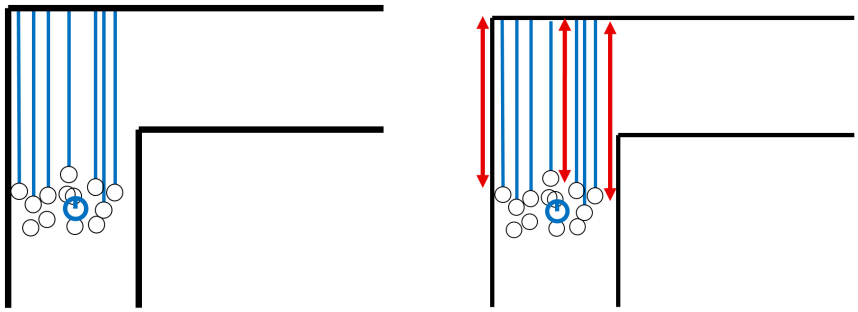
It is important to note that the NLE expects deterministic values from the measurement equation, i.e. the toolbox would calculate the likelihood using

$$p(z_k|\underline{x}_k) = \int \delta(z_k - \underline{x}_k - \underline{v}_k) f^v(\underline{v}_k) dv = f^v(z_k - \underline{x}_k) \ ,$$

where v_k is additive noise and f^v its PDF. However, both measurement models directly compute the likelihood, hence the toolbox needed to be adapted in order to work with the measurement models.

Beam range finder model The first model utilizes ray cast operations to imitate the functionality of an ultrasound sensor. Thereby, each particle performs a ray cast in the local occupancy grid map (see Figure 10).

The ray casts were implemented using Bresenham's line algorithm [Bre65]. Bresenham's algorithm determines the cells along a line in the occupancy grid map. Once an occupied grid cell has been found, the distance from the robot to this cell is computed using the Euclidean norm.



(a) (b)

Figure 10: Calculation of the expected measurement. (a) Each particle (black circles) performs a ray cast (blue) in the occupancy grid map. (b) Using Breseham's algorithm the distance (red) to an obstacle is calculated.

The beam range finder model considers four distributions for the desired likelihood. The first distribution models the case that a valid measurement has been received and is denoted by

$$p_{hit}(z_t^k|x_t,m) = \begin{cases} \eta \cdot \mathcal{N}(z_t^k; z_t^{k*}, \sigma_{hit}^2), & 0 \leq z_t^k \leq z_{max} \\ 0, & else \end{cases}.$$

The second distribution is especially designed for dynamic maps, i.e. maps which change over time. In other words, this distribution models the cases where the real measurement was less than the expected measurement

$$p_{short}(z_t^k|x_t,m) = \begin{cases} \eta \cdot \lambda_{short} \cdot \exp\{-\lambda_{short} * z_t^k\}, & 0 \leq z_t^k \leq z_t^{k*} \\ 0, & else. \end{cases}$$

Furthermore, the next distribution considers cases where measurements have been received which are greater than the maximum allowed value

$$p_{max}(z_t^k|x_t,m) = \begin{cases} 1, & z_t^k >= z_{max} \\ 0, & else. \end{cases}$$

Finally, the last distribution adds an uniform distribution for random failures

$$p_{rand}(z_t^k|x_t,m) = \begin{cases} \frac{1}{z_{max}}, & 0 \leq z_t^k \leq z_{max} \\ 0, & else. \end{cases}$$

The desired likelihood is then calculated by combining all this distributions with additional weights

$$p(z_t|x_t, m) = \begin{bmatrix} z_{hit} \\ z_{short} \\ z_{max} \\ z_{rand} \end{bmatrix}^T \cdot \begin{bmatrix} p_{hit}(z_t^k|x_t, m) \\ p_{short}(z_t^k|x_t, m) \\ p_{max}(z_t^k|x_t, m) \\ p_{rand}(z_t^k|x_t, m) \end{bmatrix}.$$

Since the measurements of an ultrasound sensor are cone-shaped, it is of practical sense to perform multiple ray cast operations within a cone. The resulting distance is then calculated as the minimum distance of all ray casts.

The main problem of this measurement model is the performance. Despite an efficient implementation of Bresenham's algorithm, the measurement model still performs insufficiently. This holds especially true for larger particle counts.

Likelihood field range finder model The second measurement model tries to eliminate the disadvantages of the *beam range finder model*. In general, the basic idea behind this model is to merely consider the expected endpoint rather than the entire path of a measurement (see figure 11).

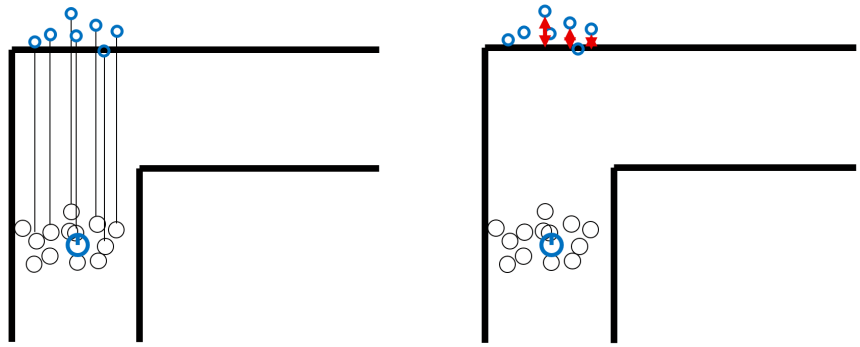


Figure 11: Calculation of the expected measurement. (a) Given a real measurement, each particle (black circles) calculates the expected endpoint (blue circles) in the occupancy grid map. (b) Distance (red) calculation based on a nearest neighbor search.

Each particle calculates the expected endpoint within its occupancy grid map using the real measurement. The calculated endpoint is then used as a initial point for a nearest neighbor search to the next obstacle. Once the nearest (minimum) distance to the next obstacle has been found the resulting likelihood can be calculated directly using the following equation

$$p(z_t^k|x_t, m) = z_{hit} \cdot \mathcal{N}(dist; 0, \sigma_{hit}) + \frac{z_{random}}{z_{max}}.$$

In addition, if the measurement falls into an unexplored area the likelihood is assumed to be

$$p(z_t^k|x_t, m) = \frac{1}{z_{max}}.$$

The advantage of the *likelihood field model* is that it performs much faster compared to the previous model. However, the *likelihood field model* has one disadvantage. Since it actually does not include the current information about obstacles, the calculation of the endpoints could result beyond the intended obstacle. Therefore, the following nearest neighbor search could compute the minimal distance to an unintended obstacle.

5.3 IMU Integration

Due to limited memory space of the Arduino it was neccesary to replace the object-oriented programming style for the Arduino sketch with a procedural style. The old sketch used approximately 96 % of program memory without any IMU support. Removing the object-orientation but adding the support of the IMU to the sketch used around 92 % of program memory space. The program used for the DMP unit and the communication between the Arduino and the IMU are provided by the MPU-6050 library for Arduino by Jeff Rowberg. [Row]

The integrated IMU (compare section 3.2) provides translational as well as rotational readings. In this work, merely the rotational readings, namely rotation about the z-axes, were used.

The robots framework features two implementations for retrieving the values of the IMU. Both implementations return the values as Euler angles in the form of *Roll Pitch Yaw* (RPY) angles within the range of -180°to 180°. The first implementation returns the absolute angles of the current state, whereas the second one returns the difference between the previous and the current state during a movement.

During the prediction step, the simulation first performs the movement and then continues with the actual prediction. This is due to the fact, that the IMU readings need to be known in advance. In each prediction step the delta for the orientation is calculated based on the mean of multiple IMU readings. The mean values of the readings are calculated using

$$\mathrm{E}\{\theta\} = atan2\left(\frac{1}{n} \cdot \sum_i sin(reading_i), \frac{1}{n} \cdot \sum_i cos(reading_i)\right).$$

5.4 Autonomous Exploration

The previous sections described utilized techniques used to estimate the robot's current position and orientation. This section describes how these techniques can be used to autonomously explore a specific environment. The crawler distinguishes between complete autonomous exploration and point-based exploration.

Point-based exploration The point-based exploration algorithm is essentially a simple *Proportional-Integral-Derivative* (PID) control algorithm. Generally speaking, the robots drives the shortest path to a given goal autonomously. The goal can either be a point in the environment set by the user or by an algorithm.

The PID algorithm computes in each iteration different error terms which are then used to create new control inputs. Given a robot's pose, the angular difference between the current location and the target location is given by

$$\phi_k = atan2((y_{goal} - y_{robot}), (x_{goal} - x_{robot})).$$

Furthermore, using this angle, the difference between the angle and target can be determined

$$error_k = angdiff(\phi_k, \theta_k),$$

where *angdiff* is a function which computes the difference between two angles within $[-180°, 180°]$.

With

$$error'_k = error_{k-1} - error_k \ ,$$
$$error_{1:k} = error_{1:k-1} + error_k$$

the desired angular velocity can be calculated by using

$$\omega_k = kP \cdot error_k + kI \cdot error_{1:k} + kD \cdot error'_k,$$

where kP, kI and kD are parameters.

The important point is that this algorithm does not consider any obstacles within the trajectory. However, the autonomous exploration does and the point-based exploration is used there to control the robot to a set of trajectory points.

Autonomous exploration For the purpose of an user-independent and user-free exploration, a fully autonomous exploration algorithm was designed. The basic idea behind this is that the robot should walk towards unknown areas and take measurements to find new free spaces as well as obstacles. Therefore, the implemented method comprises four steps: frontier detection, finding the closest frontier, finding the shortest path to this frontier and walking the crawler to the frontier (see figure 12).

Frontiers are the boarder-areas between areas known to be free and unknown locations. They are detected by calculating the gradient of a modified occupancy grid map. In this map obstacles are set to -10, free space to one and unknown space to zero. Thereby, the absolute gradient is set to be $\in (0, 1)$ only at borders where the free space changes to unknown space. All these positions are extracted and marked as frontiers. The result of a frontier detection can be seen in figure 12 (c).

The frontier which is the closest one to the crawler's position is detected using a gradient-based region growing algorithm. For this algorithm the occupancy grid map is again changed to have values equal to ten for obstacles. The robots position has the value one (being the initial region) and all other grid cells are set to zero. Finding gradients that are absolute $\in (0, 1)$ the region can be grown iteratively without growing over obstacles. This is done until the region reached a grid cell corresponding to a frontier or until the region does not grow any longer. If a frontier was found, it is returned as closest frontier. The grown region is shown in figure 12 (d).

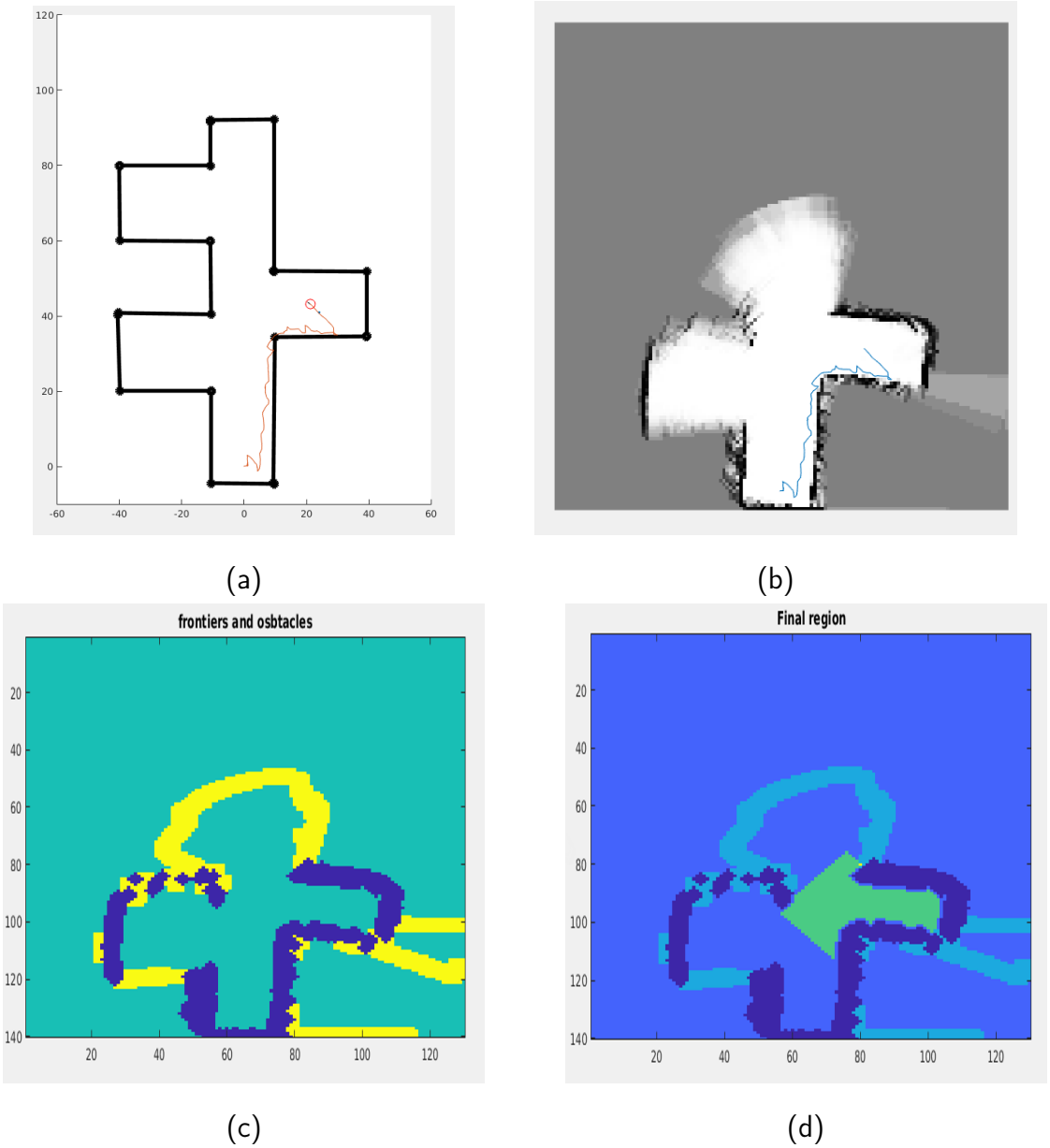


Figure 12: (a) Map used for simulation. The robot's path is indicated in red. (b) Grid-map so far generated from measurements. The estimation of the robots path is indicated in blue. (c) Detected frontiers (yellow) and obstacles (dark blue) for this step. (d) Grown region(green) and closest frontier (yellow dot). Frontiers and obstacles were expanded by a factor of two.

Since the region growing does not provide any path information from the robots position to the frontier, a path planing algorithm needs to be invoked afterwards. In this work, the implementation of the D*-algorithm in Peter Corke's Robotic Toolbox was used [Cor11; Ste94]. The algorithm computes a cost map to a given occupancy grid map. The costs represent the effort which is needed to traverse a cell, e.g. obstacle-cells have a infinite cost. Afterwards, the algorithm finds the paths with minimal costs from the goal to all free cells. The algorithm returns the path with lowest costs from the start to the goal, containing all cells which have to be traversed in the path. As the costs to transverse a free cell are equal for all cells, the path with lowest costs corresponds to the shortest path.

Finally, the previously mentioned point-based exploration is used to walk the crawler to the closest frontier. Arriving at the frontier, the robot takes new measurements and removes parts of the unknown area.

In order to avoid collision with obstacles, some derivations were added to this algorithm. Firstly, the D*-planning algorithm is only called if the closest frontier is not immediately next to the robot's position. If the robot is right next to a frontier, it just walks to it. In general, this saves computational time which would be needed for D*-planning. Secondly, an obstacle inflation was added. Therefore, first obstacles and afterwards all not interfering frontiers are expanded before the closest frontier is detected. If the robot's position is found to be inside an expanded obstacle, it is not searched for the next frontier but instead for the next path to free space. Thirdly, the crawler takes new measurements after each step taken in the path. Practically speaking, this takes more time since after each step the map as well as the position estimates need to be updated. However, it is thereby possible to receive a better estimation of the map. Finally, the crawler walk only five steps in the path resulting from the D*-planning before the whole algorithm is called again. This again results in more computation time but allows the crawler to react more adaptively to changed estimates of the map.

5.5 CSLAMv2

The already existing simulation of the previous group had to be adapted in many forms to meet the new requirements. First of all, the previous simulation did not contain any measurement model but rather approximations of it. This resulted in a lot of changes as well as improper use of the NLE toolbox by the previous group. These changes needed to be reverted and some parts even had to rewritten from scratch. The following figure 13 illustrates the taxonomy of the most important domain/model classes of CSLAMv2.

The main entry point of the application *SimCSLAMInit* defines which crawler classes and which sensors should be used in the simulation, i.e. it defines whether a simulated or a real crawler should be used. In addition, it also configures which additional sensors should be used, such as the evaluation camera or the IMU or both.

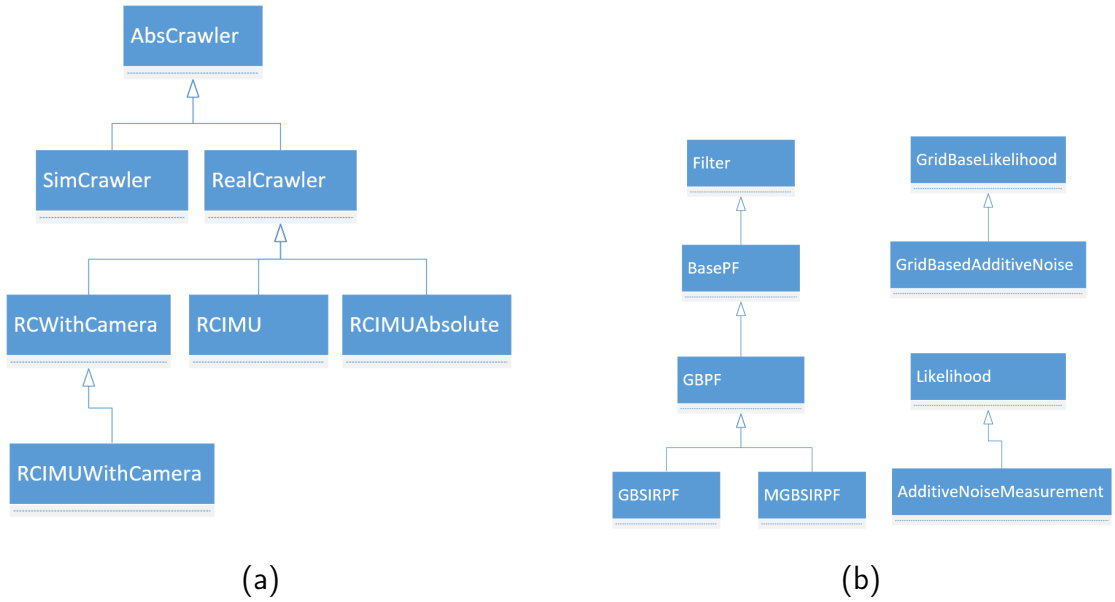


Figure 13: Class hierarchy of important classes in CSLAMv2. (a) Hierarchy of the different crawler implementations. (b) Hierarchy of the filter as well as likelihood classes.

Furthermore, *SimCSLAMInit* also creates an instance of the *CLSAM* class which acts as an application controller class. This class instantiates the *Graphical User Interface* (GUI) as well as the defined filter classes (compare figure 13).

The simulation was also improved by several aspects, e.g. in terms of usability. For example, in the previous simulation it was not possible to manually teleoperate the crawler without taking a measurement which is possible in the new version. Furthermore, the implementation of the simulation covers many different features and configurable parameters. These are discussed below.

New GUI The GUI was adapted to emphasize more on the created map as well as estimated trajectory (see figure 14).

Another reason for the new GUI was the size as well as scaling of each, the simulated world and the occupancy grid map.

Definition of filters The new simulation provides a convenient possibility for the definition of filters. The simulation allows to define an arbitrary amount of filters used during the state estimation. Each defined filter will be treated and evaluated individually as well as independently. Figure 15 shows a small run with estimations of two different filters, one with 10 and with 1000 particles. This feature can be quite useful to compare the estimations of filter with different particle count.

It is important to note that the preview of the current best estimate (compare figure 14) only shows the best estimate of the first defined filter.

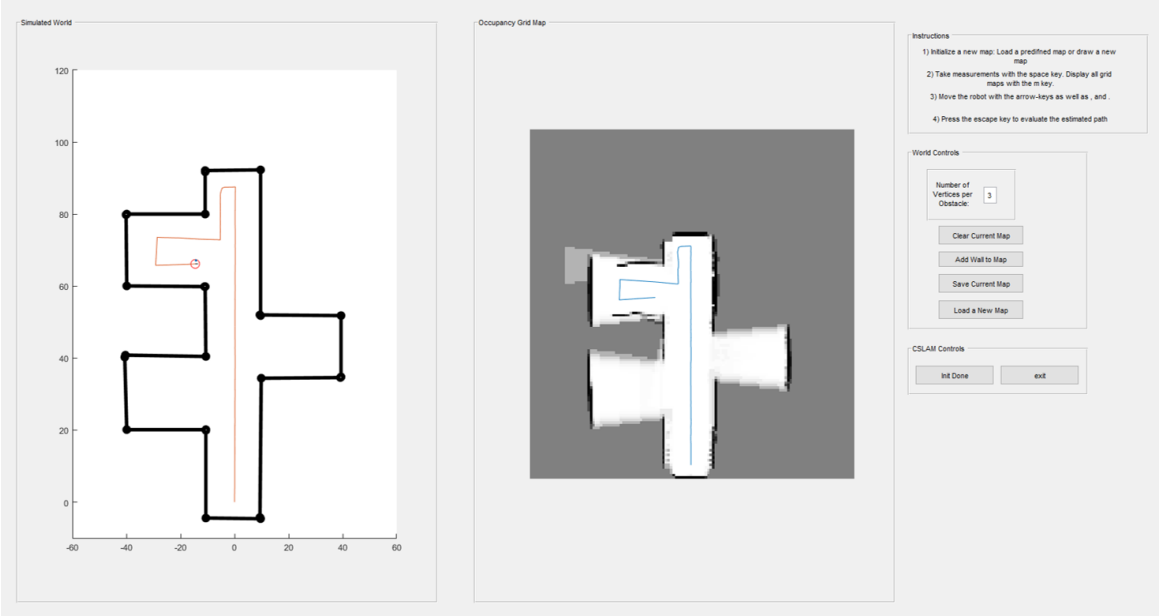


Figure 14: CSLAMv2 with a simulated world (left) and an occupancy grid (right). The simulation shows the real (red) and the current best estimate (blue) of the robot's trajectory.

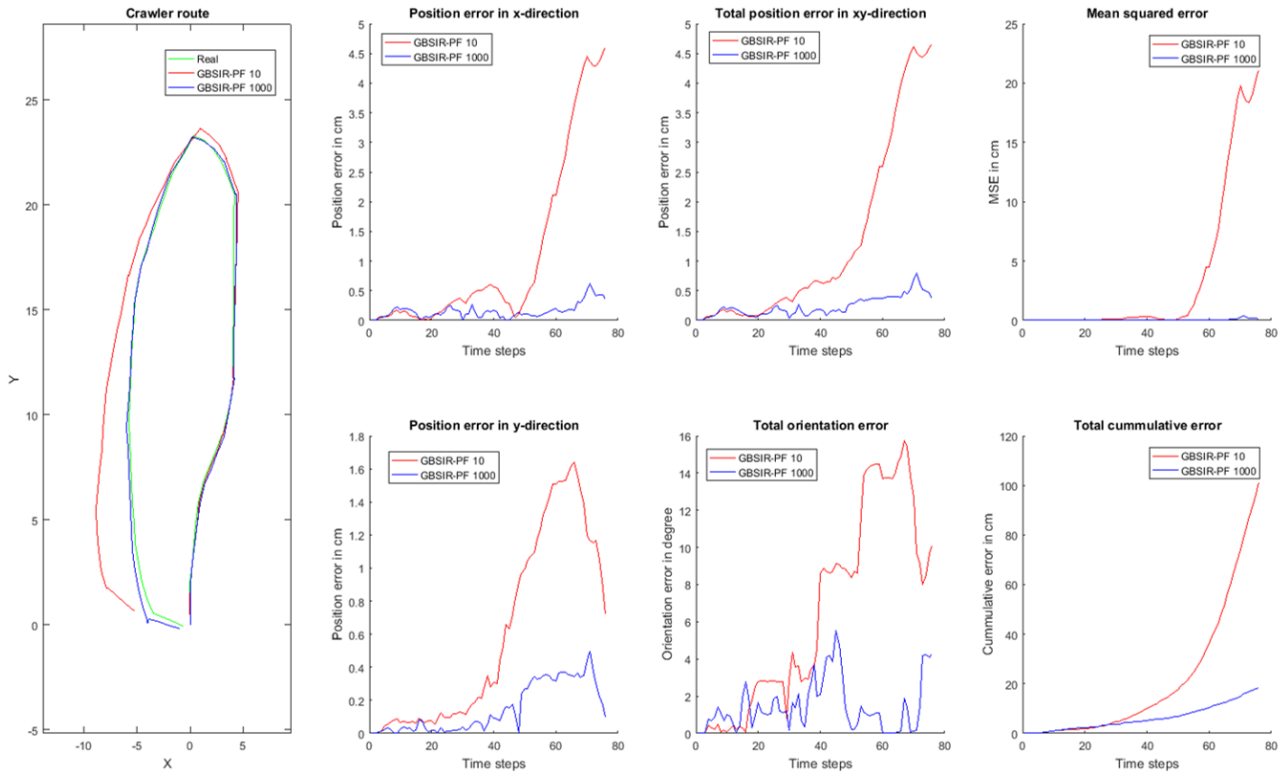


Figure 15: Evaluation of the several estimations using different filters.

New error calculation After each measurement the simulation will calculate a new estimate of the current state. This consists of the computation of $\mathrm{E}\{\underline{x}_k\}$ and $\mathrm{Cov}\{\underline{x}_k\}$ over all particles. The calculation of the expected value of θ_k needs a special treatment since it is only defined within $[0°, 360°]$

$$\mathrm{E}\{\theta_k\} = atan2(sin(\theta_k) \cdot \underline{w_k}, cos(\theta_k) \cdot \underline{w_k}).$$

These estimated points are later compared to the real values of the taken trajectory. This is done by using an integrated error calculation after the simulation took place. Besides the calculation of the total error in x- and y-direction, it shows the individual errors in each dimension, the orientation error, a cumulative sum of the errors as well as a *Mean Squared Error* (MSE) in x- and y-direction.

Definition of trajectories The simulation features two interactions with the robot in terms of movement. First, there is manual teleoperation which allows to send commands to the robot which will be executed immediately. Since in huge simulations this can be quite tedious, it is possible to define a trajectory, i.e. a list of movement commands. The robot will automatically perform these movements and will take measurements after every step.

Results and replay Each run in CSLAMv2 is stored and archived in a separate folder. This includes the calculated real as well as estimated trajectory.

In addition, CSLAMv2 will store all the movement commands and all the measurements for replaying the simulation. This can be quite helpful for replaying real simulations on a simulated crawler.

Performance improvements The old simulation contained a lot of loops and sequential code parts. This was improved by replacing some of these parts by vectorized versions of it. The vectorized version can then be compared by calculating the speed up

$$S = \frac{T_{old}}{T_{new}}.$$

5.6 Yún bridge

During the experiments, it turned out that the communication between the Arduino and the Matlab environment is very slow. Even simple commands such as retrieving the IMU readings needed more than couple of seconds. This can be quite a hassle when performing experiments with the real crawler

For the old configuration the bridge provided by Arduino was used for communication. Additionally, a REST server running on the Yún forwarded incoming requests from Matlab to the *ATMega* controller using the serial port (see figure 16). The response, e.g. the readings from IMU, was sent back to Matlab. The bridge provided by Arduino features many possibilities which are not necessary for this work (e.g. remote console access from the Arduino sketch).

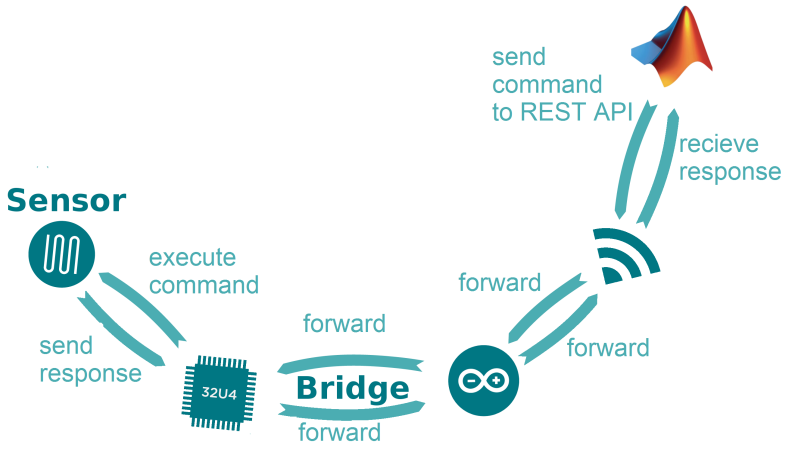


Figure 16: Communication Diagram for the Arduino Yún using the bridge. Matlab calls a REST API via the Wi-Fi. The bridge forwards the calls to the *ATmega32u4* and sends the response back to the Matlab environment [Fab+17].

In order to speed up the communication, the old bridge was disabled and a completely new bridge was developed. In order to disable the old bridge, the following line was added to */etc/inittab*

```
# ttyATH0::askfirst:/bin/ash --login
```

The new bridge runs a simple TCP server which forwards every received message to the serial port. The answer from the *ATmega32u4* will be sent back to the caller. In order to run the new bridge at start-up, it was necessary to add the following line to */etc/rc.local*, either in a console session or in the system start-up page of Luci:

```
python /usr/lib/python2.7/bridge/ownbridge/tcpipBridge.py
```

Therefore, the Arduino bridge library was no longer used which reduced the sketch size to less than 85% of the program memory.

6 Results

This chapter covers the results achieved in the course of this work. First, the results with the simulated crawler are listed and discussed. After this, the localization with the real robot as well as the techniques for retrieving them are evaluated and shown.

6.1 Simulation

This first section covers the evaluation of the simulated crawler. The simulation results are for both smaller and longer (more than 500 time steps) runs very promising. Furthermore, the overall performance of the simulation could greatly be reduced by vectorizing the software.

Likelihood field range finder model The likelihood field measurement model is very useful in terms of performance as well as simulation results. Figure 17 shows a simulation with more the 900 time steps. Since this simulation merely uses 100 particle it yields very good results.

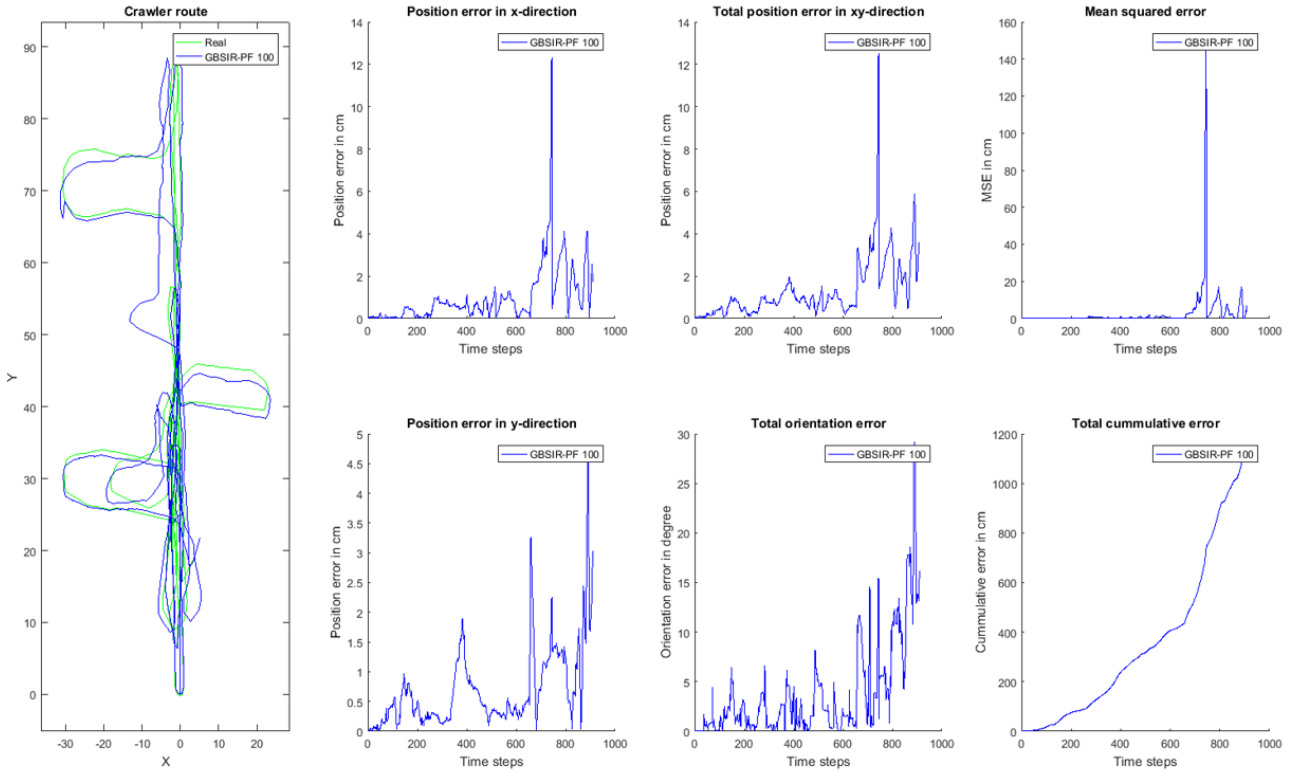


Figure 17: Large simulation with a simulated crawler using the likelihood field range finder model with only 100 particles.

This simulation provides important insights to the localization. First of all, it shows that the error in unexplored areas accumulates very fast and high.

Second, it also shows that the extensively error reduces once known areas are explored again. This can especially observed in the region around $x = -10$ and $y = 50$ of the crawler route. At

some point, when known regions are encountered, the resampling step removes bad particles and, hence, reduces the error drastically.

Beam range finder model As already mentioned in section 5.2, the beam range finder model suffers from big performance problems. Furthermore, when used in larger simulations the error of the beam range finder model seems to increase rapidly. This behavior can be seen in figure 18, the error quickly accumulates to very high values and does not decrease again.

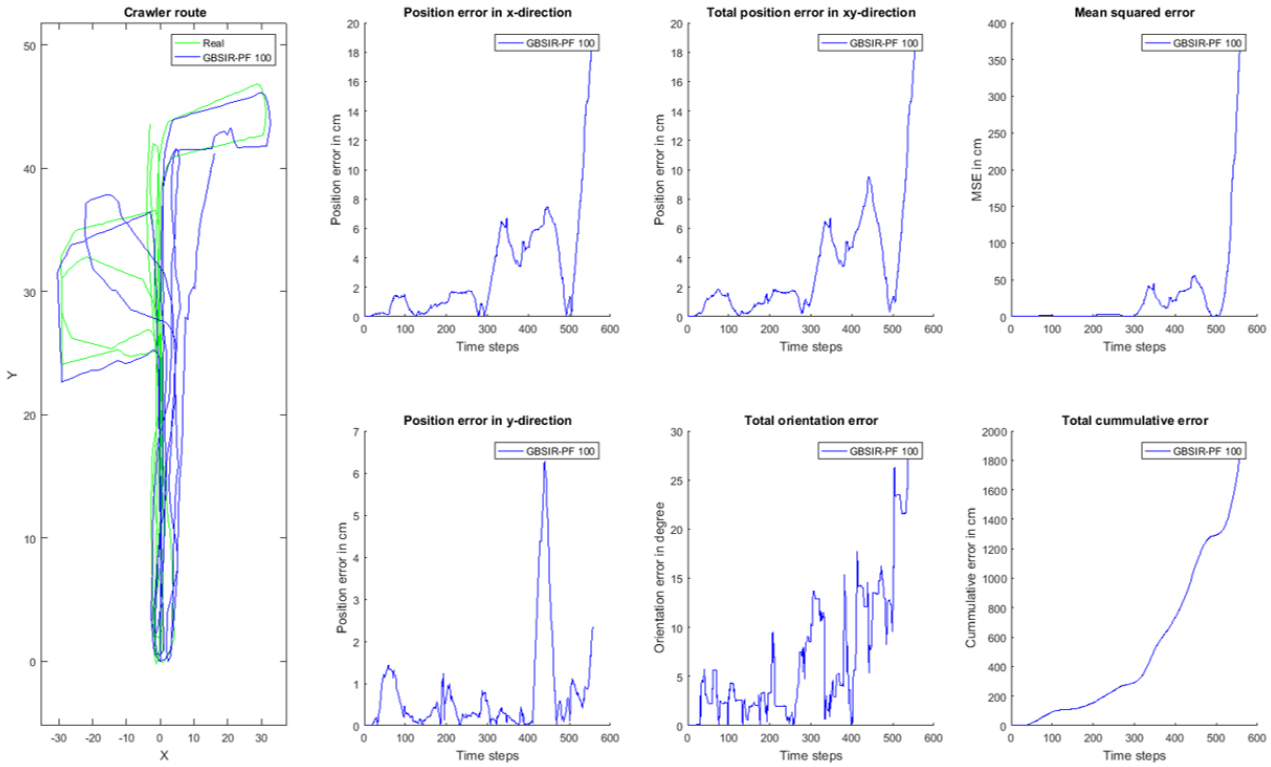


Figure 18: Large simulation with beam range finder model.

This is probably due to the fact that the measurement model works too precisely for ultrasound sensors. In this simulation the inverse measurement model shot three ray casts in different angles $\{-2.5\,°, 0\,°, 2.5\,°\}$ from the orientation of the sensor.

Performance improvements As a result of vectorizing several loops it was possible to achieve a total computation time for one single step of 5.07 seconds. The previous simulation took for the same step 15.012 seconds which results in a speedup of

$$S = \frac{15.012\,s}{5.07\,s} = 2.96 \approx 3.$$

6.2 Evaluation Techniques

In order to evaluate the estimator, a map made of cardboard was built in the course of this work (see figure 19).

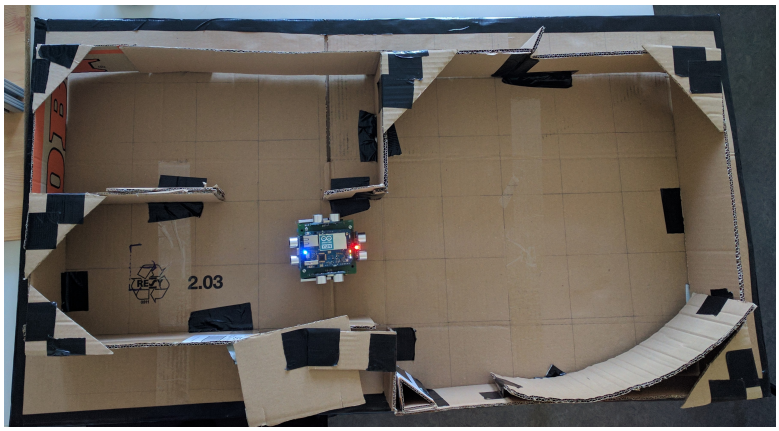


Figure 19: Evaluation scenario for the real crawler.

The dimensions of the map are approximately 118cm $\times$ 64cm. Besides the fixed walls, this map features one door and one moveable obstacle.

The crawler was evaluated using a camera hanging from the ceiling (ses figure 20) provided by the ISAS lab. The camera tracks the crawler using the LEDs which are located on the top of it. After initialization, this camera provides the real x- and y-position of the crawler. Nevertheless, the camera does not determine the robot's orientation though.

Figure 20: Used camera in the lab for evaluating the true path of the robot.

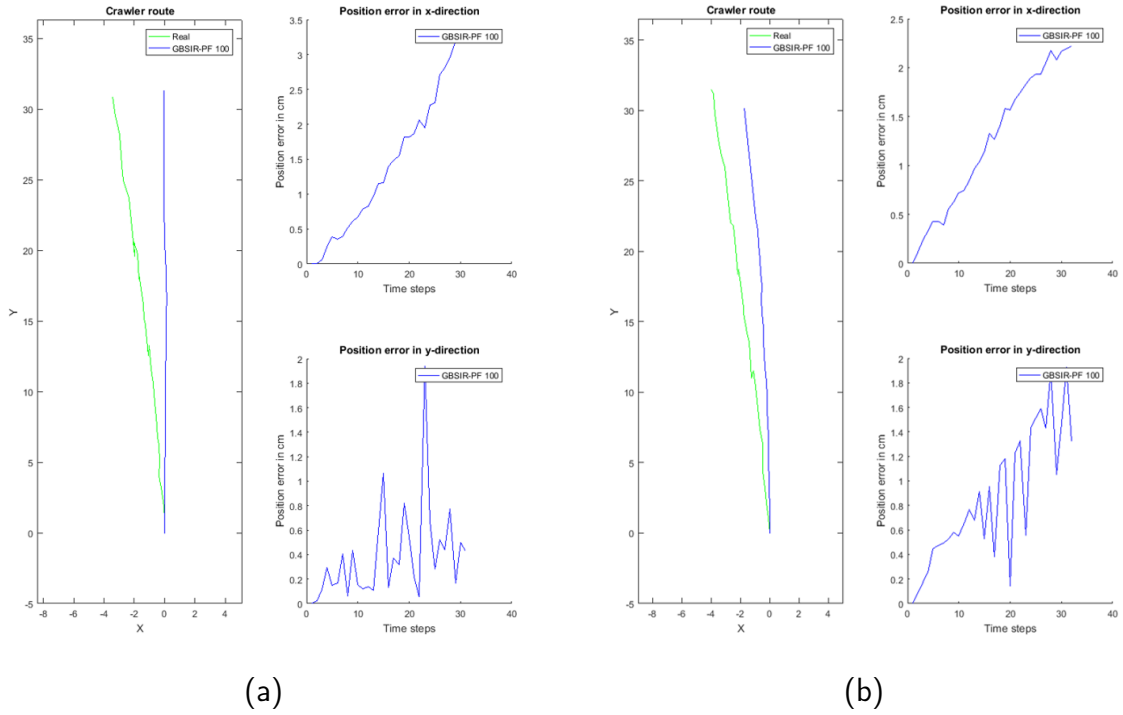
The camera will take a picture of the scene and will then compute the position of a specific maximum color, i.e. the color of the LEDs. The evaluation of the real robot using these techniques will be the topic of the following section.

6.3 Experiments

The simulation is in some terms quite different to the real robot. This is due to the fact that the real crawler contains some drift in the movement. The reason for this comes from the hardware, i.e. the constructed robot is not ideal and contains some deviations.

Initially, the big foot of the robot was blocked and thus it is was not possible to fully pull it in. After solving this issue, the robot's drift was greatly reduced.

This is why the results which were generated without utilizing the IMU are much worse compared to the result where the drift was recognized by the IMU (see figure 21). In case the estimation does not include the readings from the IMU, the PF believes that the robot went completely straight but in fact it went slightly left. This accumulates an error over time which cannot be reduced again by taking more measurements.



(a) (b)

Figure 21: Evaluation of CSLAMv2 on the real crawler. (a) Estimated route without considering the IMU. (b) Estimated route with considering the readings of the IMU.

The following experiments were all done with the IMU enabled and used for the estimation.

Bridge performance First, the performance of the crawler in terms of the communication is evaluated. Due to the use of the new bridge for communication a single retrieval of the IMU readings needs approximately less than 0.1 seconds. Compared to the old communication model, which needed approximately more than 2 seconds, this results in a speedup of

$$S > \frac{2.0\,s}{0.1\,s} = 20.$$

Beam range finder model The beam range finder model performed quite well in the beginning considering the fact that it only used 100 particles for the estimation. However, at some point the estimation went a completely different direction (compare figure 22). This experiment merely used rotational and forward movements.

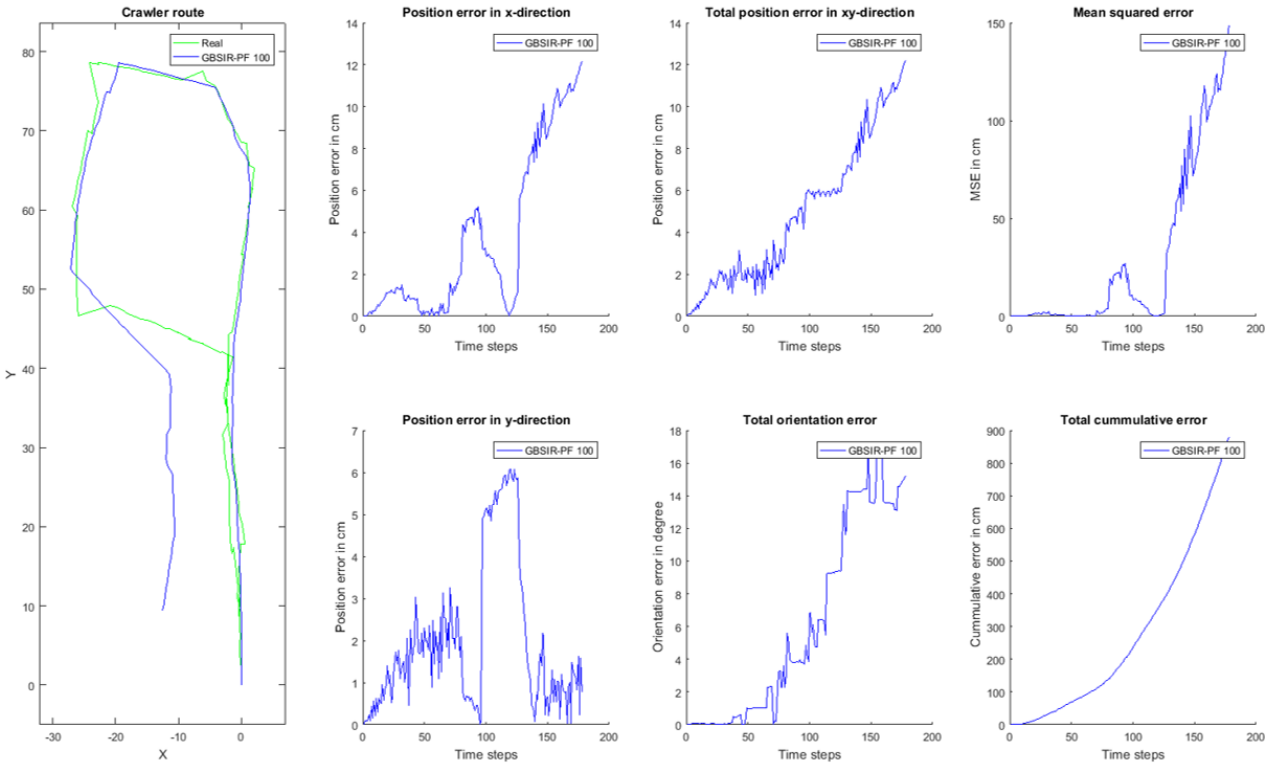


Figure 22: Evaluation of the beam range finder model on the real crawler.

Likelihood field range finder model The likelihood field range finder model yielded a good estimation for the trajectory (see figure 23). In this experiment the particle count was increased to 1500 particles.

Additionally, this experiment also made use of backward movements which suffers from a much higher drift compared to the forward movement.

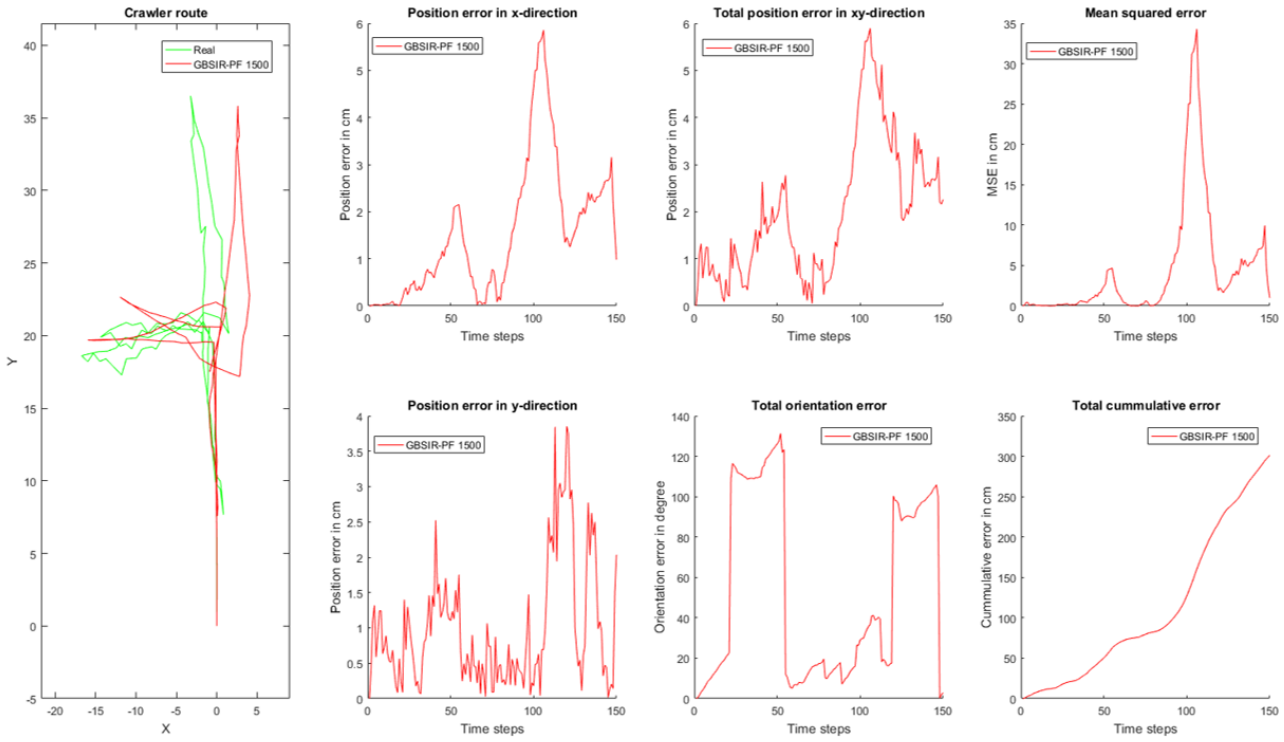


Figure 23: Evaluation of the likelihood field range finder model using the crawler.

7 Conclusion

This chapter finalizes this work. It starts by giving a general discussion of the results shown in the previous chapter and concludes by an outline of possible further work.

7.1 Discussion

The results achieved by the *likelihood field range finder model* are well enough to consider it as a basis for further research projects. The autonomous exploration is just one example of many which was now made possible by the localization. Especially the fact that only ultrasound sensors and thus distance measurements were used to localize the robot makes the results very good. The mentioned disadvantage of the *likelihood field range finder model* that it can calculate the wrong minimum distance does not really has any practical impact on the estimation.

The fact that the results are not sufficient with the *beam range finder model* most probably comes from the use of ultrasound sensors. As opposed to cone-shaped ultrasound measurement, the beam model considers the measurements to be too fine grained and thin. Therefore, certain environments, such as curvy and bended obstacles, may result in bad weights for good particles. This model is better suited for laser scans than for ultrasound measurements.

The result shown in figure 10 started quite well but then could not perform the loop closure. This probably results from the configuration of merely 100 particles.

Besides the software, the hardware, i.e. the mobile robot, works good enough now. Although, it could be argued that the execution and the software running on the robot is pretty slow and could use reconsiderations in terms of the used hardware and architecture.

During the experiments with the real crawler, it was evident that the real crawler needs much more particles in order to have a decent localization. This probably comes from the fact that the crawler does not do as many measurements as the simulated one does. Furthermore, in several cases the ultrasound measurements yield random results due to reflections and interferences.

Furthermore, all the changes which were done to the existing simulation resulted in a quite usable software. However, it still can use some improvements in some parts. Some of these potential improvements are discussed in the following chapter.

7.2 Further Work

This section deals with the current problems and potential future fixes to them.

At the moment a substantial problem with the simulation is the occupancy grid mapping. Typically, for very large simulations some obstacles start to disappear from the occupancy grid map. This results from the fact that the free space of the cone-shaped inverse sensor model absorbs the obstacles. Nevertheless, it is possible to fine-tune the parameters of the occupancy grid map to achieve better results, i.e. this can be avoided by setting the parameters to have

a more dominant occupancy probability. However, this can still not be sufficient for very large simulations. Another approach to solve this issue would be to use forward models instead of inverse sensor models [Thr03].

Another potential improvement for the simulation but especially for the occupancy grid maps is to make the maps dynamic. At the moment the occupancy grid map as well as the simulated world are completely static. In order to support larger environments these maps need to be made dynamic, i.e. their dimensions need to change over time or they need to keep track of multiple regions.

In any case, PFs are in general computationally heavy, i.e. there is always a need of performance improvements. In order to speed up, parts of the simulation could be vectorized or some parts could even be written in MEX files. MEX files are subroutines written in e.g. C/C++ and are dynamically linked.

Furthermore, the localization is also a potential part for improvements. For instance, it would be interesting to see how the combination of ultrasound with other sensors (besides the IMU) works. In terms of the used sensors, the IMU could also be replaced by an IMU which catches the drift of the robot more precisely.

Besides the improvements to the actual estimation, improving the evaluation technique would also greatly ease the realization of the experiments. For example, it would be of great advantage to estimate the robot's chassis and use this as a ground truth for the evaluation.

Finally, the autonomous exploration can be improved by adding more SLAM-specific methods to the algorithm. For example was shown by [SHB04] that the performance of SLAM-can be increased by motivating the exploration algorithm to close loops in some cases.

References

[Bre65] Jack E Bresenham. "Algorithm for computer control of a digital plotter". In: *IBM Systems journal* 4.1 (1965), pp. 25–30.

[Cor11] Peter Corke. *Robotics, vision & control: fundamental algorithms in MATLAB*. Vol. 73. Springer, 2011.

[Del+99] Frank Dellaert et al. "Monte carlo localization for mobile robots". In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 2. IEEE. 1999, pp. 1322–1328.

[Fab+17] Christoph Roth Fabian Peller et al. *Simultane Lokalisierung und Kartografierung mit dem ISAS-Crawler*. Praktikum Forschungsprojekt: Anthropomatik praktisch erfahren. 2017.

[Inv] InvenSense. *MPU-6000 and MPU-6050 Product Specification Revision 3.4*. URL: https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf.

[Mon+02] Michael Montemerlo et al. "FastSLAM: A factored solution to the simultaneous localization and mapping problem". In: *Aaai/iaai*. 2002, pp. 593–598.

[RAG04] Branko Ristic, Sanjeev Arulampalam, and Neil James Gordon. *Beyond the Kalman filter: Particle filters for tracking applications*. Artech house, 2004.

[Row] Jeff Rowberg. *MPU6050 and I2C Arduino Library*. URL: https://github.com/jrowberg/i2cdevlib/tree/master/Arduino.

[SHB04] C. Stachniss, D. Hahnel, and W. Burgard. "Exploration with active loop-closing for FastSLAM". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. 2004, 1505–1510 vol.2.

[SK16] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2016.

[Ste] Jannik Steinbring. *Nonlinear Estimation Toolbox*. URL: https://bitbucket.org/nonlinearestimation/toolbox.

[Ste94] Anthony Stentz. "Optimal and efficient path planning for partially-known environments". In: *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*. IEEE. 1994, pp. 3310–3317.

[SV00] Peter Stone and Manuela Veloso. "Multiagent systems: A survey from a machine learning perspective". In: *Autonomous Robots* 8.3 (2000), pp. 345–383.

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[Thr+02] Sebastian Thrun et al. "Robotic mapping: A survey". In: *Exploring artificial intelligence in the new millennium* 1 (2002), pp. 1–35.

[Thr03] Sebastian Thrun. "Learning occupancy grid maps with forward sensor models". In: *Autonomous robots* 15.2 (2003), pp. 111–127.

[Wei08] Florian Weißel. *Stochastische modell-prädiktive Regelung nichtlinearer Systeme*. Univ.-Verlag Karlsruhe, 2008.