Computer Engineering, Degree Project, Second Level, 30 higher education credits

Navigation on an RFID Floor

Reg. Code: Oru-Te-DT4008-D102/08

Robert Johansson

Master's Programme in Computer Engineering, 240 credits

Örebro, Sweden, Spring 2008

Supervisor: Prof. Alessandro Saffiotti
Examiner: Dr. Mathias Broxvall

NAVIGERING PÅ ETT RFID-GOLV

Örebro University
Department of Technology
SE-701 82 Örebro
Sweden

Örebro Universitet
Institutionen för teknik
701 82 Örebro

Abstract

The interest in indoor service robots is increasing day by day. Depending on the specific application, an array of issues and problems must be dealt with in order to obtain a stable and satisfactory behavior.

A major problem that applies to all types of service robots (and robots in general) is concerning the navigation. Two ingredients are necessary in order to achieve navigation; a map of the working environment, and the ability to locate in the environment. Achieving these ingredients using classical approaches in robotics is afflicted with difficult problems.

This report proposes an indoor navigation method influenced by *stigmergy*, which refers to the indirect communication that takes places between insects. A well-known example of stigmergy is the pheromone released by ants, which other ants can detect. Trails of pheromone make it possible for ants to optimize a path (i.e., find the fastest path) between two locations (e.g., the nest and the food). Stigmergy in general, can be said to, require very little information about the problem domain (compared to other approaches), when for example solving different types of optimization problems.

The proposed navigation method is called *gradient descent navigation*, and requires very little information to manage navigation. For practical reasons, pheromone has not been used for the purpose of modifying the environment. Instead, an array of RFID tags has been used, which stores a so-called *distance map*. By using a distance map (i.e., by reading data stored in the tags), a robot is able to navigate. It is also shown how a distance map can be constructed using very little information.

Sammanfattning

Intresset i servicerobotar för användning i hemmet ökar dag för dag. Beroende på det specifika tillämpningsområdet, är det nödvändigt att lösa en uppsättning av problem för att få ett stabilt och tillfredsställande beteende.

Ett betydande problem som rör alla typer av servicerobotar (och robotar generellt) är rörande navigationen. Två ingredienser är nödvändiga för att uppnå navigation; en karta över arbetsmiljön, och förmågan att lokalisera i miljön. Att uppnå dessa ingredienser genom klassiska metoder i robotiken är behäftat med svåra problem.

Denna rapport föreslår en navigeringsmetod som är influerad av *stigmergi*, vilket avser den indirekta kommunikation som sker mellan insekter. Ett välkänt exempel på stigmergi är feromon som avsöndras av myror, vilket andra myror kan detektera. Spår av feromon gör det möjligt för myror att optimera en gångstig (dvs., hitta den snabbaste vägen) mellan två platser (t.ex. boet och maten). Generellt kan man säga att stigmergi kräver väldigt lite information om problemdomänen (jämfört med andra metoder), då t.ex. olika typer av optimeringsproblem ska lösas.

Den föreslagna navigeringsmetoden kallas *gradient descent-navigering*, och kräver mycket lite information för att klara navigering. Av praktiska skäl har feromon inte använts för syftet att modifiera miljön. Istället har en samling RFID-taggar använts, som lagrar en så kallad *avståndskarta*. Genom att använda en avståndskarta (dvs., genom att läsa data sparat i taggarna), är det möjligt för en robot att navigera. Det visas även hur en avståndskarta kan konstrueras med mycket lite information.

Contents

1 Introduction **1**

1.1 AASS . . . 1
1.2 PEIS Ecology . . . 1
1.3 Project Description . . . 2
1.3.1 Motivation . . . 2
1.3.2 Goals . . . 2
1.3.3 Limitations . . . 3
1.3.4 Preliminary Work . . . 3
1.4 Disposition . . . 4

2 Basics of RFID tags **5**

2.1 Overall Description . . . 5
2.2 Types of RFID tags . . . 6
2.2.1 Passive RFID tags . . . 6
2.2.2 Active RFID tags . . . 6
2.3 Coupling Methods . . . 7
2.3.1 Inductive Coupling . . . 7

3 Development Environment **9**

3.1 Hardware Setup . . . 9
3.1.1 PEIS Home . . . 9
3.1.2 RFID Reader Hardware . . . 10
3.1.3 Mobile Robot . . . 10
3.2 Software Setup . . . 11
3.2.1 Player . . . 11
3.2.2 Stage . . . 12
3.2.3 PEIS Kernel . . . 13
3.2.4 PEIS RFID . . . 13
3.2.5 KickOff . . . 14
3.2.6 Tag Viewer . . . 14

4 Fuzzy Rule-Based Control **15**
4.1 Top-Level Description . . . 15
4.2 Fuzzification . . . 16
4.3 Rule-Base . . . 17
4.4 Inference . . . 18
4.5 Defuzzification . . . 18

5 Gradient Descent Navigation **21**
5.1 Optimization . . . 21
5.1.1 Gradient Descent(Ascent) in Continuous Functions . . . 22
5.1.2 Gradient Descent(Ascent) in Discrete Functions . . . 24
5.2 Distance Maps . . . 25
5.2.1 Definition of Distance Map . . . 25
5.2.2 Storage of Distance Maps . . . 27
5.3 Executing a Distance Map . . . 28
5.3.1 Hardware Considerations . . . 28
5.3.2 Determining the Lowest Cost Direction . . . 29
5.3.3 Avoiding Obstacles . . . 29

6 Distance Map Building **31**
6.1 Off-line Distance Map Building . . . 31
6.1.1 Map Building by Breadth-First Search . . . 32
6.2 On-line Distance Map Building . . . 32
6.2.1 Proposed Building Method . . . 33

7 Testing and Evaluation **35**
7.1 Gradient Descent Navigation . . . 35
7.1.1 Data Collection Strategy . . . 35
7.1.2 Test 1 – Simple Path In Both Directions . . . 36
7.1.3 Test 2 – Difficult Path In Both Directions . . . 37
7.1.4 Test 3 – Moderately Difficult Path In One Direction . . . 39
7.1.5 Conclusions . . . 41
7.2 Distance Map Building . . . 42
7.2.1 Difference Between Distance Maps . . . 43
7.2.2 Expectations . . . 44
7.2.3 Test 1 – Building in a small space . . . 44
7.2.4 Test 2 – Building in the full space . . . 45
7.2.5 Conclusions . . . 49

8 Discussion and Conclusions **51**
8.1 Work Overview . . . 51
8.2 Conclusions . . . 52

8.3 Future work 53

9 Acknowledgments **57**

References **59**

A Logged Gradient Descent Paths **61**

Chapter 1

Introduction

This report describes the result of a degree project, entitled "Navigation on an RFID Floor", that was performed during the spring of 2008 at AASS, Örebro University, as a spin-off of the ongoing PEIS Ecology project.

1.1 AASS

AASS, an abbreviation for "Applied Autonomous Sensor System", is a research center located at the Department of Technology at Örebro University. AASS pursues internationally acknowledged research within several scientific disciplines, among others, computer technology and artificial intelligence [1]. The research philosophy at AASS aims at combining the scientific disciplines needed for the construction of autonomous sensor systems.

A broad and general definition of autonomous sensor systems could be: independent systems, often some sort of robot, that takes impressions from the surrounding environment, and then performs actions based on those impressions.

AASS consists of three different research laboratories; Intelligent Control, Learning Systems, and Mobile Robotics. This degree project has been performed at the Mobile Robotics Lab, under the supervision of Prof. Alessandro Saffiotti, who is also the head of the lab.

1.2 PEIS Ecology

One of several ongoing research projects at AASS is called PEIS Ecology (Ecology of Physically Embedded Intelligent Systems). The PEIS Ecology project aims at combining two visions within the field of robotics; (1) to create intelligent assistants intended to exist in people's homes and workplaces to perform different kinds of physical tasks, (2) to create a network of intelligent devices with the ability to help people cognitively (e.g., provide information). The goal with the PEIS Ecology project is ultimately to improve the life of people in the future.

Many current approaches of obtaining (1) and (2) is to construct an isolated unit, preferably as human-like as possible, with extraordinary physical and cognitive skills [5]. PEIS tries to tackle this problem by instead using a group of (less complex) cooperating

units that can together solve common goals. A unit is simply referred to as a PEIS, and can be objects ranging from simple light bulbs to complex robots.

For a more detailed description of the aims and goals of the PEIS Ecology Project, see [14], which is the first paper published on the subject. [15] could also be of interest since it is a summary of achievements during the initial years of the project.

1.3 Project Description

1.3.1 Motivation

The interest in service robots is increasing strongly. During construction of such a robot, many difficult problems must be solved in order to obtain satisfactory behavior. One of the primary problems when building a service robot is its navigation capabilities. Navigation in this context is defined as *moving a robot from an arbitrary start position to a predefined goal position*. It is important that navigation is accurate and that its accuracy does not decline by time.

To achieve navigation in the classic approach, robots often use a discrete representation of the working environment called map, often stored in memory. One of the major problems is the construction of such a map, especially when sonar is used; issues like the field of view of the sonar sensors and specular reflections should be considered. Another issue is maintaining the accuracy of the position estimation as navigation is performed; e.g., only relying on odometry will undoubtedly fail. This is because odometry inherently accumulates errors due systematic and non-systematic errors that can never be eliminated, only lowered [8]. These two issues are very difficult to solve accurately by a robot alone without any external help, and thus it is problematic to achieve navigation in a stable way.

A non-classic approach of doing navigation is to devise an approach influenced by *stigmergy*. Stigmergy refers to the indirect communication that takes places between insects by modification of the environment. A well-known example of stigmergy is the pheromone trail left out by an ant, which can be detected and followed by other ants . Such trails actually make it possible for ants to optimize the path (i.e., find the fastest path) between two locations (e.g., the nest and the food source) [9].

Stigmergy in general, requires very little information about the problem domain compared to other approaches when for example solving different types of optimization problems. As indicated a few lines above: it is interesting that ants, without any prior information (e.g., distances) can find the shortest path between two points. A classic approach of doing the same optimization as the ants would be to utilize a map of the environment and performed a search using some well-known graph search algorithm, like breadth-first search.

1.3.2 Goals

The goal with this project is to propose a new method of doing indoor robot navigation influenced by stigmergy. An important aspect is to minimize the required information about the problem domain, i.e., supply the navigation robot with a map should not be necessary. Another aspect is to remove difficult problems like position estimation. Hopefully, the proposed method will not suffer from as difficult problems as those described above.

For practical reasons, pheromone or any other chemical substance will not be used. Instead, an array of RFID tags will be used for the purpose of modifying the environment. Each tag will store the cost of traveling to a predefined goal, and a robot will perform navigation by continuously looking at tag data. This way of doing navigation will be referred to as *gradient descent navigation*. An array of cost data will from here on be referred to as a *distance map*.

Giving the RFID tags cost data will be done by an on-line method, i.e., by physically moving the robot around, as opposed to constructing the distance map off-line (in-memory) which would require a map of the working environment. The method for giving the goal is simply to place the robot at the goal before distance map building is started.

To conclude, the two major goals with this project are to

- propose a method for doing gradient descent navigation
- propose a method for on-line distance map building

1.3.3 Limitations

Although this report is comprehensive and detailed in many aspects, some issues are ignored and not mentioned at all (as the observant reader might will detect). This has been done in an attempt to keep the report from getting too detailed, or as simplifying assumptions. For completeness, the most major (ignored) issues are mentioned below.

- In preparation for an on-line distance map build, it is necessary to clear the associated cost of each tag in the working environment. In Section 6.2.1, where the proposed on-line distance map building method is given, it is not explained how this can be done. Depending how the cost of each tag is stored (e.g., physically in the tags), the process of clearing the costs might or might not be an issue. When physically storing in the tags, this is an issue, since each tag must be accessed and cleared. During this project a virtual RFID tag array was used (see Section 5.2.2), where this issue is not present at all. Still, this is mentioned as a limitation as it is probably is desired in real applications to physically store in the tags.

- The working environment is assumed to be static, i.e., never change, e.g., obstructions is neither removed nor added. This implies that a distance map never becomes inaccurate or outdated. Thus, this report does not deal with the question of how a distance map can be updated to reflect changes in the working environment.

1.3.4 Preliminary Work

A student project at AASS has been performed on the topic gradient descent navigation, entitled "Target Reaching by Gradient Descent on an RFID Floor".

The purpose of the project consisted primarily of installation, i.e., set up certain necessary hardware and software. Test gradient descent navigation using an off-line constructed distance map was also included. Due to many hardware issues, the project never reached the point where gradient descent was implemented and performed.

1.4 Disposition

Chapter 1 provides an introduction to this project, giving necessary preliminaries in order to understand why this degree project has been performed.

Chapter 2 provides a brief introduction about RFID tags, including different types of tags. A full chapter has been dedicated to RFID tags due to their importance.

Chapter 3 gives a summary of different tools that was used during the course of this project, i.e., different types of hardware and software.

Chapter 4 gives an introduction on fuzzy control.

Chapter 5 suggests how navigation can be done using a stigmergic approach, which is referred to as *gradient descent navigation*. This chapter starts out by shortly explaining *gradient descent(ascent)* – a gradient informed optimization algorithm. Moreover, how to perform gradient descent in discrete functions is shown. Further, a deeper definition of distance maps is given, stating some important properties. It is finally shown how a distance map can be physically executed by a robot.

Chapter 6 deals with the construction of a distance map. It is explained how off-line building is done by using graph search algorithms. Also, a proposal of how on-line building can be performed is given, i.e., by physically moving a mobile robot around.

Chapter 7 contains evaluations of gradient descent navigation and on-line distance map building.

Chapter 8 discusses the achieved results, possible future work.

Chapter 2

Basics of RFID tags

In the introduction chapter, it was mentioned that so called RFID tags have been used as the means of modifying the environment. This chapter aims at giving a brief introduction on this topic. All information was fetched from [4] and [17].

2.1 Overall Description

RFID is an abbreviation for *Radio Frequency IDentification*, and is a term referring to a type of system where communication between the constituent components is realized using radio frequency (RF) or magnetic field variations. That is, communication is done wireless and without the requirement of a clear view between the communicating components. Such a system consists (at a bare minimum) of a *tag* and a *reader*. The purpose of the tag (also referred to as a *transponder*) is usually for identification of objects, hence the "identification" in RFID. That is, the tag is physically attached to the object that is to be identified, and acts like an identification device. The reader is the device that can detect the presence of tags, and read information stored on the tags. After a successful read, the reader can then notify some other system about the presence of a tag, and forward fetched data for further processing.

Tags often incorporate a so-called UID, an abbreviation for Unique IDentifier. In this context, it refers to a collection of bits located in the memory areas of a tag, used to uniquely identify it. More advanced (and thus also more expensive) tags also offers a general-purpose, non-volatile, user-programmable memory, so that information about the attached object can be stored in the tag itself, and not in a separate storage system. In a separate storage system, the information about attached objects are stored by the UID.

It should be emphasized that a tag works both as a transmitter and a receiver; a tag intercepts an RF signal, and then transmit a response. This also applies to the reader. Moreover, both readers and tags make use of antennas to manage communication.

Although RFID tags are usually seen as a way of doing identification, there is nothing that says it must be used for this purpose only; a RFID tag can be seen as a general-purpose, wireless-accessed storage device, usable for many more applications than just identification.

2.2 Types of RFID tags

A common way of categorizing different types of RFID tags is by how they obtain power. This characteristic is actually a major factor for the size, cost and longevity of a tag; *passive tags* obtain their power by means of utilizing the RF signal sent out from the reader; *active tags* use an internal power source in the form of a battery cell. Traditionally, tags that make use of an on-board power source for some of its function, and the RF signal of the reader for other functions, have been referred to as active as well. A more recent term for this type of tags is *semi-passive*. This section will only focus on passive and active tags.

2.2.1 Passive RFID tags

The most notable feature of passive tags is that they completely lack an internal power source; they are powered solely by an electric current induced in the antenna of the tag by the reader's RF signal. Often, this induced current is just enough to power up the tag, calculate the response, and finally transmit the response back to the reader. This is the reason why passive tags have limited capabilities compared to active tags.

Passive tags obtain power from the reader's RF signal by a phenomenon known as the *near field*. It occurs when the magnetic part of the RF signal is powerful enough to induce a current in a coil. Induction is based on the property that when a coil is placed in an alternating magnetic field, energy is drawn from the field, resulting in a voltage in the coil. This means that in a system of passive tags, both readers and tags use coils, and not conventional radio antennas. As suggested by the name, the near field is only available in vicinity close to the reader's antenna; the strength of the magnetic field decline very fast.

The advantage of passive tags (compared the active tags) are generally low price, small size, and a simpler manufacturing process. Apparent disadvantages include a small memory capacity, and limited calculation abilities. Usually, the limited range is also considered as a disadvantage. In this project, the limited range of passive tags has actually been an advantage.

2.2.2 Active RFID tags

The major characteristic of active tags is that they use an internal power source, i.e, batteries, and with this follow larger possibilities in almost every respect compared to passive tags. For example, since a battery is used to power the circuitry, it is possible to transmit and receive without having to be powered by the near field of the reader's antenna. More importantly, the limited communication range imposed by the near field does not apply.

The advantages of active tags (compared the passive tags) are generally higher memory capacity and higher calculation abilities (perhaps encryption of transmitted data to prevent eavesdropping). Moreover, communication is often more reliable, due to the more reliable power source. Disadvantages are generally higher price, and larger in size (possible bulky).

2.3 Coupling Methods

The term *coupling method* refers to the way in which a reader and a tag communicate or send power. There are several determining factors for the coupling method to use. One important factory is the power source of the tag. It should be mentioned that the coupling method directly determines the communication range between the tag and the reader.

In this section, we satisfied describing only one coupling method − inductive coupling. This because it is the method that applies most to this project; tags are to be detected close to the reader's antenna. Note that there are several more coupling methods.

2.3.1 Inductive Coupling

Tags that make use of inductive coupling manage communication by using the near field, and are most commonly passive tags. That is, passive tags use the near field for the purposes of both maintain power and communication. How power is maintained was briefly explained in the section about passive tags.

In inductive coupling, communication is achieved by turning a resistor on and off in the tag. This causes fluctuations in the magnetic field created by the reader's antenna, resulting in changes in the voltage, which is interpreted by the reader as a binary number. This procedure is refereed to as load modulation.

Chapter 3

Development Environment

This chapter aim at presenting different types of hardware and software used during the course of the project.

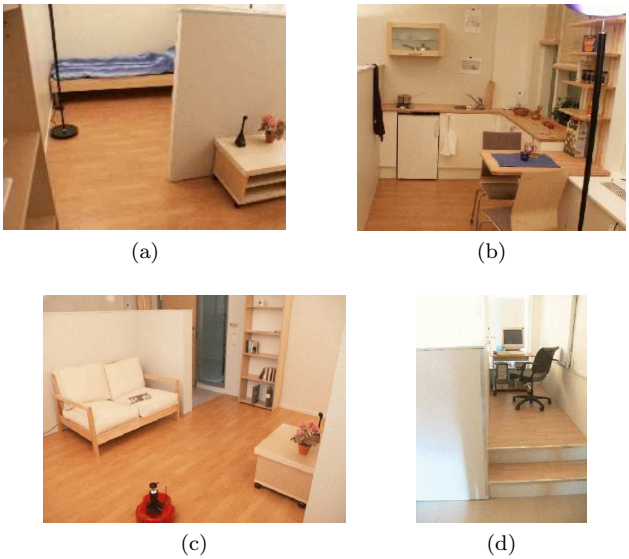
3.1 Hardware Setup

3.1.1 PEIS Home

To test theories and concepts within the PEIS Ecology Project, an experimental environment has been constructed called the PEIS Home, which at a first glimpse looks like a normal apartment. It consists of a living room, kitchen, and a bedroom. The walls between these spaces are only 1.40 meters high, so an observer with more ease can overlook the whole apartment. Next to the apartment, there is an elevated observation deck where a PC has been placed where experiments can be conducted from. The size of the apartment including the observation deck is approximately 7 meters by 4 meters, which is a fairly small space. Several unusual objects (compared to a regular apartment) can be found in PEIS Home, including mobile robots, cameras, and RFID tags located underneath the floor. See Figure 3.1 for photos of PEIS Home.

As indicated in the introduction chapter is the RFID tags of particular interest. A more detailed description of the RFID tags in PEIS Home is therefore appropriate. Underneath the parquet floor, there has been placed approximately 400 RFID tags. The RFID tags have been positioned to form a hexagonal lattice. This means that neighboring tags are equally distanced from each other, and that each tag has six neighbors. Neighboring tags in PEIS Home are spaced at a distance of circa 26 centimeters. See Figure 3.2 for an explanatory image where a bird's-eye view of the apartment is shown together with the layout of the RFID tags. As seen in Figure 3.2, some RFID tags appear to be missing. This is because only working tags are shown (i.e., working when the inventory of tags was made).

The actual tags used in PEIS Home are called *Tag-it HF-I Plus*, developed by Texas Instruments. In size, they are similar to a common credit card, disregarded from the thickness; while a credit card is typically one millimeter thick, these tags are thinner than 0.1 millimeters at the antenna areas, and close to 0.35 millimeters at the chip area. When it comes to memory capabilities, it has several memory areas (as RFID tags in general), which includes a user programmable area of 2 kibibyte, and a 64 bit vendor-programmed read-only area containing the UID. Further, the Tag-it tags are of



(a) (b)

(c) (d)

Figure 3.1: A few photos taken from inside the PEIS Home apartment: (a) depict the bedroom. (b) shows the kitchen area. (c) shows the living room. Also note the mobile robot placed on the floor; the base of the robot is of the same type as on the robot used in this degree project. (The black object mounted on the base is an omni-directional camera). And finally, (d), which shows the observation area. It is located next to the living room. All photos were fetched from [3] and [14].

type passive, and powered by the inductive coupling principle. See [11] for a data sheet of the Tag-it tags. It contains more technical details.

3.1.2 RFID Reader Hardware

The Feig ISC-MR100A RFID tag reader was the piece of hardware used for the purpose of reading/writing data from/to tags in the working environment. This particular hardware is intended for use with passive RFID tags. Furthermore, communication and powering are achieved according to the inductive coupling principle.

Furthermore, it was found by testing that, this reader used with the tags located in PEIS Home allowed up to three tags to be in range simultaneously. However, usually one or two tags were in range at the same time.

3.1.3 Mobile Robot

As mentioned in the introduction chapter, the use of a mobile robot is essential. For this purpose the Centibot has been used, which is developed by MobileRobots (formerly ActivMedia). The Centibot platform consists of the Amigobot mobile robot and an on-board PC (simply referred to as the PC from now on). See Figure 3.3. Note that the figure does not perfectly match the actual Centibot used; missing is (1), the RFID tag reader (described in 3.1.2), (2), a wireless bridge, and (3), the RFID antenna. (1) was placed on top of the PC, and connected to it by the USB interface through an RS-232 to USB converter cable. The connection between the Amigobot and the PC was done

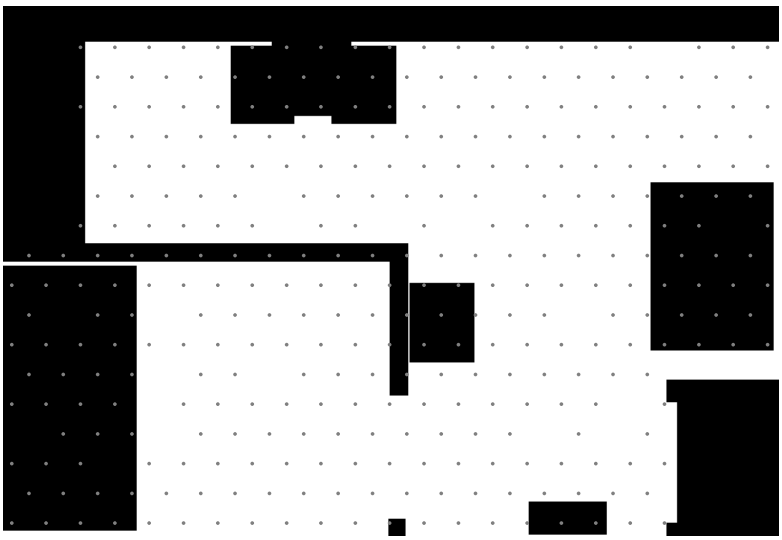


Figure 3.2: A bird's-eye view of the PEIS Home. Note that only the main apartment is shown; the observation deck has been excluded. Black represent objects that occlude the floor, or where the parquet floor is not present. A gray dot represents a RFID tag. For example, the bottom-left black rectangular object represents the bed. Note how the placement of the RFID tags form a hexagonal lattice.

by the same approach. (3) was placed underneath the robot, and slightly in front of the wheel axis, to allow a wider area of RFID tags to be covered during rotation of the robot. The purpose of (2) was to allow wireless access to the PC and connected hardware.

The technical specifications of the AmigoBot is as follows [12]:

- In size, it is 33 cm long, 28 cm wide, and 13 cm high (not including body clearance).
- Differential-drive steering, and a rear caster to prevent the base from tipping over.
- Maximum translational speed of 0.75 meters/sec, and rotational speed of 300 degrees/sec.
- Eight range-finding sonar sensors; four are placed in the front, two on each side, and two in the rear.

3.2 Software Setup

3.2.1 Player

Programming robots can be a difficult task, especially in distributed and multi-robot settings [10]. In respond to this, several open-source softwares has been developed within the Player Project. Player is one of those softwares, and it provides common interfaces for controlling and managing a variety of different robotic and sensor systems. Since the start of the Player Project in 1999 at University of Southern California, Player has become a de facto standard in the open-source robotics community [16].

Figure 3.3: The Centibot platform. It consists of the Amigobot mobile robot, and a fairly small, on-board PC. Image was fetched from [2].

In most common settings, Player functions as a TCP-based device server, which multiple clients can connect to. It is also possible for a client to connect to multiple Player servers. Other settings are possible, e.g., a monolithic approach where an application communicates directly with devices (without any network layer).

Architecturally, Player takes influence from operating system design in the sense that device interfaces are separated from device drivers [10]. For example, one of the most used device interfaces in Player is *position2d*, used for controlling the base of planar mobile robots. The interface itself cannot do anything, it is simply a specification. One implementation of *position2d* is the driver *p2os*, which provides the means to control robots developed by MobileRobots.

Since only device interfaces are known to users of a Player server, a Player client developed for one device will work on another (different) device, as long as drivers exist for both devices implementing interfaces used by the client.

3.2.2 Stage

Stage is another application within the Player Project. It is a planar, low-fidelity simulator software intended to be used in conjunction with Player. A major advantage with Stage is that it performs its work underneath the Player middleware. In other words, Stage resides in the bottom of a software stack consisting of Stage, Player, and Player clients. The advantage of this is clear; Player clients will work (without recompiling) on both simulated and physical robots.

Stage supports a lot of the interfaces in Player, including *position2d* (for controlling 2D robots), and *sonar* (for sonar systems), which was the interfaces most used during this project. The working environment of a simulated robot is supplied by simply using an image (e.g., stored in a .png file). Figure 3.4 shows a screenshot of Stage in action, where an image of PEIS Home is used.

In this project, Stage has been particularly used for testing of obstacle avoidance, implemented by a so-called fuzzy controller. More information about this will follow.

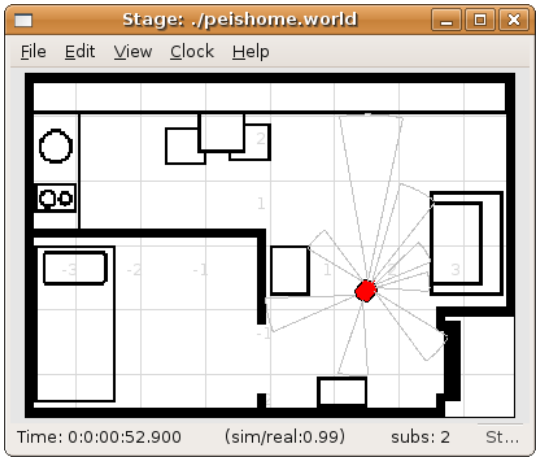


Figure 3.4: An instance of Stage, loaded with an image showing the working environment of PEIS Home.

3.2.3 PEIS Kernel

When devising a system consisting of many non-uniform devices, difficult problems arise, often due to interface problems. Most often this is managed by some middleware, providing a uniform interface for all involved devices. Player, as discussed in 3.2.1 is an example of this, providing a common interface for handling of various robotic systems.

Due to that a collection of collaborating PEIS units per definition will be non-uniform, communication between such units could become an issue. To resolve this, a uniform communication model has developed within the PEIS Project, and implemented as a middleware referred to as the PEIS Kernel. It provides a distributed tuple-space together with event mechanisms. An instance of the PEIS Kernel, running inside a physical PEIS, can register tuples in its local tuple-space. Such a set-up is called a PEIS Component. (A PEIS can have several running PEIS Components). PEIS Components can subscribe to non-local tuples, and through the event mechanism, subscribers are notified when subscribed tuples changes. Thus it is not necessary for subscribers to employ polling in order to detected changes in subscribed data, and thus decreasing the risk of network congestion. For an in-depth technical description of the PEIS Kernel, see [6].

PEIS Kernel is an open-source software released under the LGPL license. It can be downloaded for free from the homepage of the PEIS Ecology Project[1].

3.2.4 PEIS RFID

PEIS RFID is an application developed at AASS, providing a simple interface for the RFID hardware described in 3.1.2. The application periodically polls the RFID hardware to detect changes of tags in the reader's range. When a change occurs, a tuple simply named *tags* located in the local tuple-space is updated to reflect the UIDs of tags currently in the reader's range.

The version of PEIS RFID used did not support updating of RFID tag data. This however, did not pose any problem throughout this project since a virtual RFID tag

[1] http://www.aass.oru.se/~peis/

array was implemented and used, that only relies on the UID of a tag. This will be explained in more detail in subsequent chapters (see Section 5.2.2).

3.2.5 KickOff

KickOff is an application written by Prof. Alessandro Saffiotti, intended to interface with a robot through Player. The application offers a graphical interface for controlling the robot, map building by use of sonar, a simple fuzzy rule-based controller, and other things. It was originally developed to be used in teaching. In this project, it has been used to decrease the amount of time spent (or remove) on various programming tasks, like familiarization with the Player framework, implementation of a fuzzy controller, etc.

3.2.6 Tag Viewer

Tag Viewer is a small application developed specifically for this project. See Figure 3.5 for a screenshot. This application implements all algorithms and methods described in this report. Moreover, it incorporates use of the software described in this chapter. For example, it interfaces and controls a mobile robot by accessing a Player server, and it uses PEIS RFID to gather the UID of tags in range of the RFID reader, and so on. Although not a requirement, it has also been given a graphical user interface (GUI). The most important part of the GUI is that it allows visualization of RFID tag data, which Figure 3.5 gives an example of. The meaning of all colors will be explained in following chapters. The blue color can be explained already at this point; it is used to denote a tag that is currently within the reader's range.

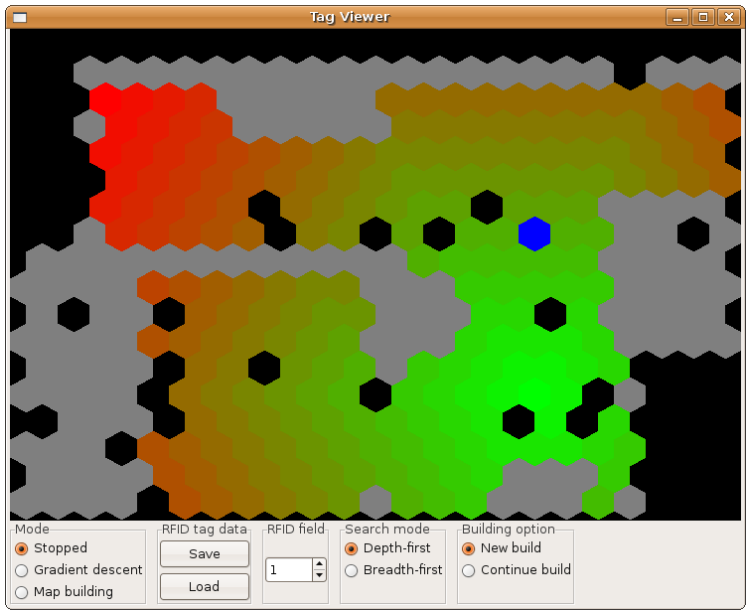


Figure 3.5: Tag Viewer – the control application written and used during the project.

Chapter 4

Fuzzy Rule-Based Control

As described in later chapters, a so-called fuzzy controller has been used to solve a small but important issue when it comes to robot navigation in an unknown environment; obstacle avoidance. This chapter aim at briefly introduce fuzzy control. All information was fetched from [13].

4.1 Top-Level Description

Fuzzy control is an interesting way to deal with control of a system. This is particular the case when a linear model[1] of the system is not available, but where heuristic information[2] is. For example, heuristic information might be gained from an operator of a machine of some sort. Very trivial examples also exist; we humans have learned by experience that walking into objects is undesirable, so we intuitively use knowledge similar to "if there is an obstacle too close to the left, then go somewhere to the right".

In the process of constructing a fuzzy controller, the first step is to fetch heuristic knowledge, which could be done by consulting an expert of the system to be controlled. Note that there are many ways this could be done; e.g., the heuristic knowledge could also be based on a mathematical analysis of the system. Regardless of how the knowledge is gathered, next step is to insert it into the fuzzy controller. At this point, the fuzzy controller will try to recreate the actions of the expert by infer, or draw conclusions, from the knowledge.

In this context, heuristic information is often linguistic descriptions expressed in some human-readable form, like English. These linguistic descriptions are often referred to as *rules*, and are commonly of the form "If-Then", just like the previously given example.

Most often, a fuzzy controller is interconnected with the controlled system by the classical closed-loop setting; the fuzzy controller affects the system trough looking at system output. Other approaches are possible (depending on type of system to control), like open-loop control. Figure 4.1 shows a fuzzy controller working in closed-loop mode. As Figure 4.1 indicates, a fuzzy controller consists of four major parts:

[1] In this context, linear model refers to a description of the mechanics behind a system, utilizing only a collection of linear equations. Linear models are often characterized by predictability and simplicity. Nonlinear systems, on the other hand, are often much less understandable.

[2] Heuristic information is knowledge gained from working-experience of a system, that could help in solving a problem.

1. *A fuzzification stage*, where input variables are converted into so called *linguistic values*. More about this below.
2. *A collection of rules* (often called rule-base), which contains knowledge in form of rules, describing how to control the system given the output from the fuzzification stage.
3. *A inference mechanism*, which performs evaluation of rules. The output from this step describes (in an abstract way) what output to send to the system.
4. *A defuzzification stage*, where the abstract output from the inference mechanism is converted into crisp values, directly usable by the system.

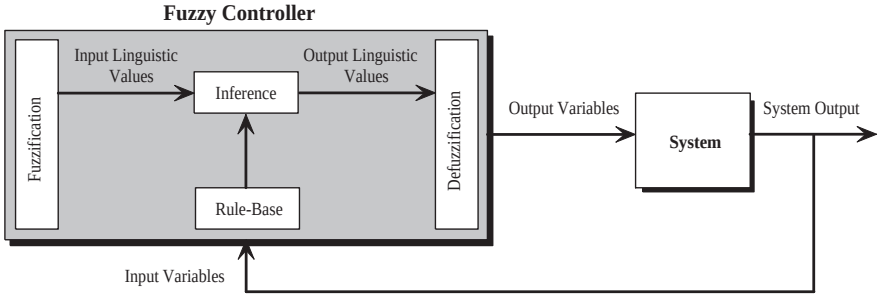


Figure 4.1: The constituent parts of a fuzzy controller − and how such a controller work in closed-loop mode with some system.

4.2 Fuzzification

The fuzzification step is important, since it performs a conversion of crisp input values into a form usable by the fuzzy controller. In particular, it transforms vague descriptions like "too close", "too low", etc., into well-defined mathematical descriptions. In case of obstacle avoidance, a crisp value is much likely to be a distance measurement.

The first step in fuzzification is the creation of *linguistic variables*, in which simply controller inputs are named. Changing of name does not affect the result of the fuzzy controller; it is mere a construction to ease development of a controller.

The second step is to devise linguistic values for each one of the linguistic variables; while controller inputs are often real numbers (e.g., distance measurement, voltage, etc.), linguistic variables take on linguistic values (also referred to as fuzzy predicates). To do things a bit less abstract, consider obstacle avoidance yet again. In order to do good obstacle avoidance of e.g., a robot, it is likely distance measurements is at least to be taken on both sides (left and right), and in front, and then supplied to the fuzzy controller. Denote these inputs by x_i, where $i = 1, 2, 3$, and $x_i \in [0, \mathbb{R}^+]$. Now let $\tilde{x}_i$ denote the linguistic variable of x_i. Let's refer $\tilde{x}_1$, $\tilde{x}_2$, and $\tilde{x}_3$ to *measurement left*, *measurement right*, and *measurement front*, respectively. Further, let $\tilde{x}_i^j$ denote a linguistic value of $\tilde{x}_i$, where $j = 1, 2, ..., n_i$. n_i is the amount of linguistic values for $\tilde{x}_i$. For simplicity, assume we are making use of only two values for each one of the variables; *obstacle close to left*, and *obstacle fairly close to left* for variable *measurement left*, and so on for the other two variables.

The third step is to precisely define what the linguistic values mean. This is obligatory, because it is else would be impossible for a controller to know what e.g., *obstacle fairly close left* means. For this purpose, *fuzzy sets* are used, which is a generalization of the notion of a set. In fuzzy set theory, as opposed to basic sets, members can have *degree of membership*. This is represented by a real number in the range $[0.0, 1.0]$, where 0.0 means no membership whatsoever, and 1.0 is full membership. So, to define the meaning of a linguistic variable $\tilde{x}_i^j$, a membership function $\mu_i^j : X \rightarrow [0.0, 1.0]$ is used, where X is the domain of $\tilde{x}_i$. Figure 4.2 gives an example of how the membership functions for *obstacle close to left* and *obstacle fairly close to left* could look like. Note that the look of the functions (i.e., triangular) is mere a suggestion; arbitrary look of a membership function is possible. The KickOff fuzzy controller, which was the controller used in this project, uses triangular membership functions.

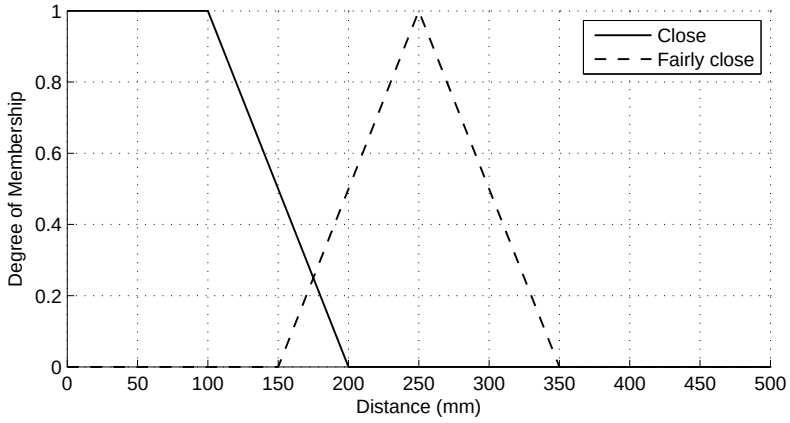


Figure 4.2: Example membership functions that could be used to define linguistic values for for the purpose of achieving obstacle avoidance.

4.3 Rule-Base

The following is examples of possible rules for obstacle avoidance utilizing some of the linguistic values described in the previous section.

IF obstacle close to left AND NOT obstacle close to right THEN turn right fast
IF obstacle close to right AND NOT obstacle close to left THEN turn left fast
IF obstacle close to left AND obstacle close to right THEN turn none

As seen above, a rule consists of an antecedent part (IF), and a consequent part (THEN). The consequents of rules are abstract descriptions of what to input the system. In this particular case, rotational guidelines of how to avoid obstacles. Actually, consequents are linguistic values – just like those the fuzzification step produces. The linguistic variable for the consequents should probably be referred to as *rotational speed*.

4.4 Inference

Inference refers to evaluating all rules in the rule-base. The aim is to decide the degree of membership for the rule consequents, which is done by evaluating corresponding antecedent parts. But as can be seen in the previous section, it is possible to create Boolean expressions of input linguistic values. If this had not been the case, the evaluation process would have been simple; the degree of membership for a consequent would have equal that one of the only linguistic value in the antecedent part. This has been solved by extending the standard set operators *complement*, *intersection*, and *union*, which is often named as NOT, AND, OR, respectively, when used in a fuzzy controller, as in the rules in the previous section.

Fuzzy set operators are defined in terms of membership functions; let μ_i^a and μ_i^b be two membership functions defined over the domain of $\tilde{x}_i$. Further, let A and B denote the fuzzy sets described by membership functions μ_i^a and μ_i^b, respectively. Then,

$$\mu_{A^C}(x) = 1.0 - \mu_i^a(x) \qquad \text{(Complement)}$$

$$\mu_{A \cap B}(x) = min(\mu_i^a(x), \mu_i^b(x)) \qquad \text{(Intersection)}$$

$$\mu_{A \cup B}(x) = max(\mu_i^a(x), \mu_i^b(x)) \qquad \text{(Union)}$$

where $x \in$ domain of $\tilde{x}_i$. For example, consider the first rule in the previous section. Assume that *obstacle close to left* is 0.6, and *obstacle close to right* is 0.7. Then the second operand of AND will evaluate to 0.3, due to complement of *obstacle close to right*. The whole antecedent will evaluate to 0.3, since $min(0.7, 0.3)$ equals 0.3. So, the resulting degree of membership for *turn right fast* is 0.3. For completeness, second and the third rule would evaluate to 0.4 and 0.6, respectively.

4.5 Defuzzification

Defuzzification refers to converting the conclusions drawn by the inference step into crisp values. Generally speaking, it implies to somehow use the output linguistic values ($\tilde{y}_i^j$, where $j = 1, 2, ..., n_i$) for each output linguistic variable ($\tilde{y}_i$, where $i = 1, 2, ..., n$), to produce crisp values. A very popular way to do this is by the so called *center of gravity* method (COG), which assumes that a membership function $\mu_i^j(y)$ is defined for each linguistic value $\tilde{y}_i^j$. It also assumes that a "best representing" crisp value b_i^j is given for each $\tilde{y}_i^j$, e.g., a crisp value that peaks $\mu_i^j(y)$. For the obstacle avoidance example given previously, a "best representing" crisp value for "turn none" is intuitively 0. In case of a membership function with a symmetric triangular look, a good choice of this value is the middle of the triangle.

The actual conversion to crisp value (y_i) is performed by the following

$$y_i = \frac{\sum_j b_i^j \int min(a_i^j, \mu_i^j(y))}{\sum_j \int min(a_i^j, \mu_i^j(y))} \tag{4.1}$$

where a_i^j is the degree of membership for $\tilde{y}_i^j$. What the integral does is to calculate the area under a "chopped of" version of $\mu_i^j(y)$ (i.e., chopped at $y = a_i^j$). This has the effect when a a_i^j increase, so does the corresponding integral, resulting in giving b_i^j more

"weight", or importance. Figure 4.3 shows possible "chopped of" membership functions for the obstacle avoidance example, using the a_i^j values calculated previously.

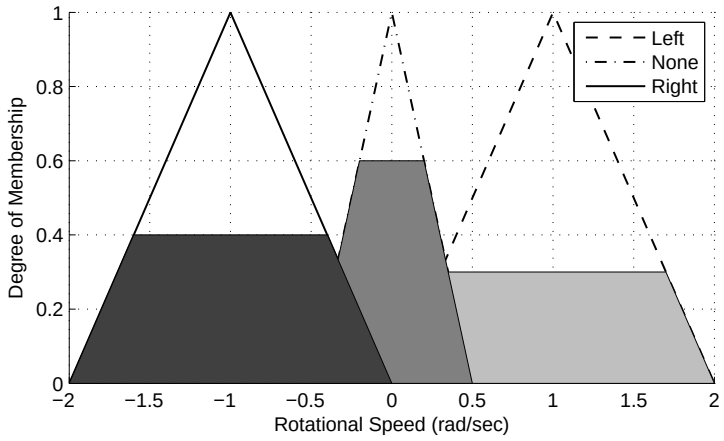


Figure 4.3: Possible output membership functions for the obstacle avoidance example.

There also exist simpler defuzzification methods, that does not require membership functions to be defined for each $\tilde{y}_i^j$. One common is called center-average defuzzification (also called COG of singletons), defines as

$$y_i = \frac{\sum_j b_i^j a_i^j}{\sum_j a_i^j} \tag{4.2}$$

where a_i^j is the same as before. This actual, is the defuzzification approach used by the KickOff fuzzy controller. Let's apply this method to the calculations in the previous section. First, define *turn left fast*, *turn right fast*, and *turn none* to 1 rad/sec, -1 rad/sec, and 0 rad/sec, respectively. The crisp value is calculated by equation (4.2) as

$$\textit{rotational speed} = \frac{1 \cdot 0.4 - 1 \cdot 0.3 + 0 \cdot 0.6}{0.4 + 0.3 + 0.6} \approx 0.08$$

so the output crisp value is circa 0.08 rad/sec, which means a slight rotation to the left. This result is sound, because the example indicated an obstacle closer to the right, than to the left.

Chapter 5

Gradient Descent Navigation

This chapter explains the proposed gradient descent navigation method. But not only this, the concept *gradient descent(ascent)* is explained, which is a relatively simple optimization method for multi-variable functions. Gradient descent(ascent) for both continuous and discrete functions is shown. Optimization algorithms are discussed for the reason that the proposed navigation method closely reminds of optimization, but in a "physical" sense. Further, distance maps are defined more closely; a vague description was given in the introduction chapter.

5.1 Optimization

Optimization means to under some well-defined conditions find the best solution to a given problem. The "best solution" is found in a set of feasible solutions. The problem at hand can be to maximize the profit in a company, calculate the shortest route between two cities, minimize the amount of material necessary for a tin can given the desired capacity, etc. In mathematics, this type of problems often translates into determining the maximum or minimum of some function f.

Very often, f will have the following definition

$$f : D \to \mathbb{R}$$

where D is a subset of $\mathbb{R}^n$, i.e., $D \subseteq \mathbb{R}^n$. Using this definition, then maximization can be defined as finding an element $\mathbf{x}^* \in D$, such that $\forall \mathbf{x} \in D : f(\mathbf{x}^*) \geq f(\mathbf{x})$. The definition of minimization is very similar, since only the inequality needs to be changed; minimization is to find an element $\mathbf{x}^* \in D$, such that $\forall \mathbf{x} \in D : f(\mathbf{x}^*) \leq f(\mathbf{x})$.

For twice-differentiable functions f, optimization is usually performed by looking at derivate information. In single-variable functions $f(x)$, where $x \in \mathbb{R}$, extreme points can be found by finding roots of the first derivative $\frac{df}{dx}$. To determine the type of found extreme points, the second derivative $\frac{d^2f}{dx^2}$ is used. This is basic calculus. If $f(x)$ is not differentiable, or for any reason derivatives are not feasible to use, then numerical approaches are possible. An example of such an algorithm is *Golden-Section Search*.

In multi-variable functions $f(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2, ..., x_n)$ and $n \geq 2$, it is possible to find extreme points through the *gradient* of f. The gradient of f is denoted by ∇f, and defined as

$$\nabla f(\mathbf{x}) = \left(\frac{df}{dx_1}, \frac{df}{dx_2}, ..., \frac{df}{dx_n} \right) \tag{5.1}$$

As seen in (5.1), the gradient is a vector consisting of the first-order partial derivatives of f. It can be shown that if $\nabla f(\mathbf{x}^*) = \mathbf{0}$, then $\mathbf{x}^*$ is an extreme point of f. The type of an extreme point $\mathbf{x}^*$ can be determined by the so called *Hessian*, which is a $n \times n$ matrix containing second-order partial derivatives of f. As with single-variable functions, there exist numerical approaches for finding the optimum in multi-variable functions. The simplest one is *random search*, which repeatedly evaluates f at randomly selected $\mathbf{x}$. More sophisticated numerical approaches include gradient informed methods, like *gradient descent(ascent)*, discussed below.

5.1.1 Gradient Descent(Ascent) in Continuous Functions

Consider the problem of climbing to the summit of a hill. For the sake of the discussion, assume blindfolds are being worn. An intuitive strategy would be to determine the maximum slope in the starting position, and then walking in the direction of maximum slope. It is obvious that, unless the starting position is on a ridge leading directly to the summit, this approach will not work. A slightly cleverer method is to walk a certain distance defined by a *step size*, then reevaluate the maximum slope, and again walk a step size in the maximum slope direction. This is repeated until the summit is reached (i.e., the slope in all directions is zero, or close to zero). This actually, is a description of the gradient ascent method. Gradient descent is very similar − the only difference is that minimums are approached instead of maximums, so descent uses the direction of the minimum slope.

More mathematically, gradient descent(ascent) is applied as follows. (Only the two-variable case is shown. This is partly due to it simplifies the notation a bit, partly that the two-dimensional case is most applicable for this project, partly that extension to the n-variable case is simple). Let f be defined as $f : \mathbb{R}^2 \rightarrow \mathbb{R}$. Then gradient descent(ascent) is a process of iteratively determine points $\mathbf{x}_0$, $\mathbf{x}_1$, ... that hopefully converge towards an optimum $\mathbf{x}^*$ (i.e., $\nabla f(\mathbf{x}^*) = \mathbf{0}$). In gradient ascent, each iteration consists of the following.

1. Let $\mathbf{x}_k$ be the current position. Determine the direction $\mathbf{v}_k$ in which the slope of f is greatest (i.e., most positive) in the point $\mathbf{x}_k$.
2. Determine an appropriate step size h_k, and choose next position as $\mathbf{x}_{k+1} = \mathbf{x}_k + h_k \cdot \mathbf{v}_k$.

Still, the above description does not reveal exactly how $\mathbf{v}_k$ is determined. Actually, it is possible to show that $\mathbf{v}_k = \nabla f(\mathbf{x}_k)$. Similarly, $\mathbf{v}_k = -\nabla f(\mathbf{x}_k)$ when gradient descent is performed. This implies that the gradient gives the direction of maximum slope.

The selection of a step size h_k can be done by simply picking a constant value. The risks associated with this are clear; selecting a too big step size could result in non-convergence, i.e., the optimum is repeatedly missed. (Consider walking one kilometer in each iteration, while the hill is a few meters in size). Too small step size could possibly be inefficient. A good way (optimal way) of selecting the step size is by finding an extreme point to

$$g(h_k) = f(\mathbf{x}_k + h_k \cdot \mathbf{v}_k) \tag{5.2}$$

Finding a maximum for (5.2) results in a step size h_k that will maximize f along the maximum slope direction. This is obviously usable for the gradient ascent method. In words, this means "walk in the direction of maximum slop until the hill start to slope downwards". Similarly, finding a minimum is usable for the gradient descent method. This way of doing gradient descent(ascent) is called *optimal gradient descent(ascent)*.

Figure 5.1 shows graphically how fifteen iterations of the optimal gradient ascent algorithm behaves when applied to the function

$$f(x, y) = 2.25xy + 1.75y - 1.5x^2 - 2y^2 \tag{5.3}$$

at start coordinate $\mathbf{x}_0 = (1, 1)$. In the figure, note that both the surface and contour lines (i.e., a contour map) are shown in the same plot. Further, it is possible to show that (5.3) has a maximum at $(x, y) = (63/111, 84/111) \approx (0.57, 0.76)$, which is marked out by a dot. The perpendicular relationship between two connecting line-segments is due to the use of (5.2).

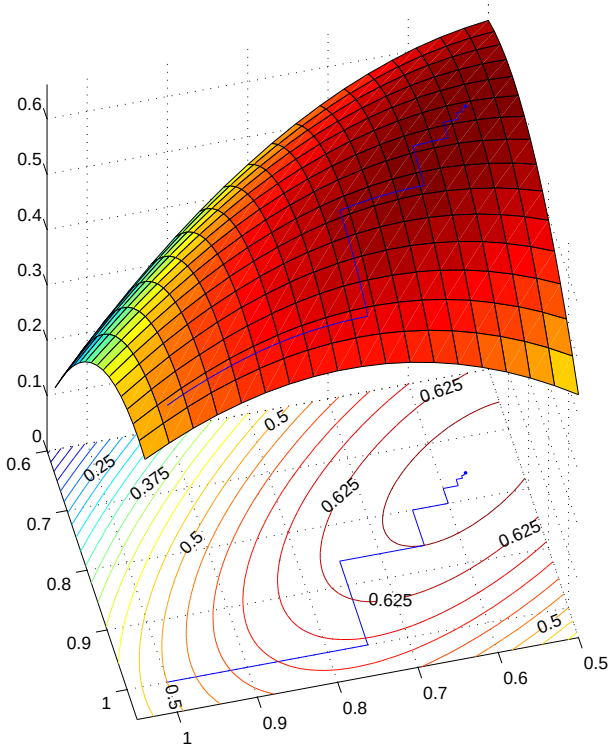


Figure 5.1: The result of applying fifteen iterations of the gradient ascent algorithm to (5.3), with a step size determined by finding a maximum to (5.2), i.e., optimal gradient ascent. The start coordinate was set to be $\mathbf{x}_0 = (1, 1)$. A blue line segment indicates one iteration of gradient ascent, and the actual path if hill climbing were to be performed physically. The optimum is located at approximately (0.57, 0.76).

5.1.2 Gradient Descent(Ascent) in Discrete Functions

Doing gradient descent(ascent) in a discrete function is not very difficult, and does not require as much theory compared to continuous functions. One intuitive way of finding the global optimum of a discrete function f is to iterate over the whole domain of f. Although this approach is simple, it might not be desirable in some cases. One case is where the domain of f is very big; iterating over very many elements could possibly be too time consuming. Another case is when it is expensive (by some definition) to change between elements in the domain of f. For example, consider the task of physically climb a discrete hill (e.g., imagine a set of neighboring plateaus, that are square-shaped, and with common length and width). Since the iteration approach is used to get the summit, this means the height of each plateau must be determined. Then, when the optimum is known, movement to the summit can start. But this is expensive since it involves a lot of movement between plateaus, and would be especially costly in case of a large domain. Clearly, if the search was to be performed by a computer, with the hill represented by some structure in memory, e.g., a two-dimensional array, then movement between two arbitrary elements is done in constant time.

Discrete Descent(Ascent) Method 1

A better approach of doing discrete gradient descent(ascent) is by looking at neighbors of the current position. Let us refer this algorithm as *discrete descent(ascent) method 1*. First, define f as

$$f : D \to \mathbb{R}$$

where $D \subseteq \mathbb{Z}^n$. Then, let the set $S = \{\mathbf{n}_1, \mathbf{n}_2, ..., \mathbf{n}_m : \mathbf{n}_1, \mathbf{n}_2, ..., \mathbf{n}_m \in \mathbb{Z}^n\}$, denote the m neighbors, expressed in relative coordinates, for an arbitrary $\mathbf{x} \in D$. Gradient ascent can then be performed by repeating the steps below, which will determine points $\mathbf{x}_0$, $\mathbf{x}_1$, ... that hopefully will converge towards the optimum $\mathbf{x}^*$.

1. Let $\mathbf{x}_k$ be the current position. Let $S_v = \{(\mathbf{x}_k + \mathbf{n}) \in D : \mathbf{n} \in S\}$ be the set of all valid neighbors of $\mathbf{x}_k$. Stop searching if $\mathbf{x}_k$ is the optimum, i.e., when $\forall \mathbf{x} \in S_v : f(\mathbf{x}_k) \geq f(\mathbf{x})$.

2. Determine the neighbor of $\mathbf{x}_k$ with greatest value. That is, find an $\mathbf{x}_{max} \in S_v$, so that $\forall \mathbf{x} \in S_v : f(\mathbf{x}_{max}) \geq f(\mathbf{x})$. Choose next position as $\mathbf{x}_{k+1} = \mathbf{x}_{max}$.

Discrete Descent(Ascent) Method 2

The above algorithm is less expensive than the approach of iterating over all elements of D. Still, since the neighbors of each visited $x \in D$ needs to be examined, this approach could be expensive, especially when doing a physical search. Another, better approach can be devised. Let us refer to it as *discrete descent(ascent) method 2*. Of the discrete methods described here, this one resembles the optimal gradient descent(ascent) method most. For gradient ascent, it is performed by doing the following steps (using the same definitions as in *discrete descent(ascent) method 1*).

1. Let $\mathbf{x}_k$ be the current position. Stop searching if $\mathbf{x}_k$ is the optimum, i.e., when $\forall \mathbf{x} \in S_v : f(\mathbf{x}_k) \geq f(\mathbf{x})$.

2. Determine the neighbor of $\mathbf{x}_k$ with greatest value. That is, find an $\mathbf{x}_{max} \in S_v$, so that $\forall \mathbf{x} \in S_v : f(\mathbf{x}_{max}) \geq f(\mathbf{x})$. Then, calculate the gradient as $\mathbf{v}_k = (\mathbf{x}_{max} - \mathbf{x}_k$, which evaluates to the relative coordinate of $\mathbf{x}_{max}$ for $\mathbf{x}_k$. If two or more neighbors can be associated with the greatest value, a good solution is to add there (normalized) relative coordinate vectors together, resulting in an "average" gradient. Another method is to simply select one of the neighbors, ignoring the others.

3. Follow the gradient, and evaluate f at each entered $\mathbf{x} \in D$. When a decrease of f is detected, stop following the gradient, and go to step 1.

To conclude this section, Figure 5.2 shows the result of applying the two described methods on a discretized version of (5.3). Note that function has been discretized onto a square grid; this has been done out of visualization simplicity. In the actual work, the focus is solely only hexagonal grids (hexagonal lattices). See Section 5.2.1 for a deeper explanation of this.

5.2 Distance Maps

In the introduction chapter, the purpose of a distance map was briefly explained. It was also mentioned that a distance map contains "cost data" to some predefined goal. In this section, a more detailed definition of a distance map is given.

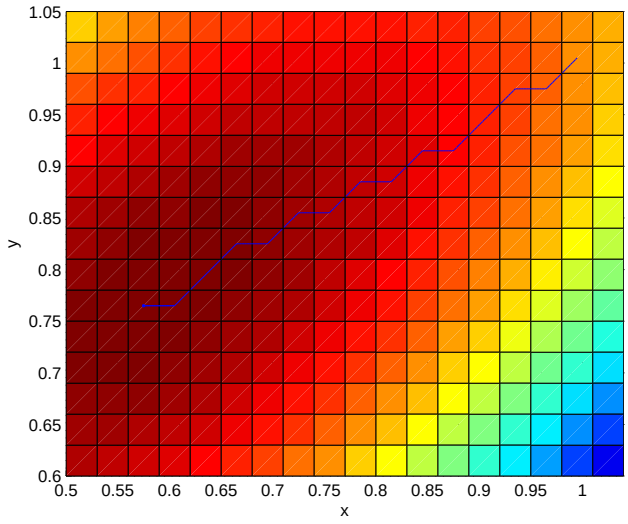
5.2.1 Definition of Distance Map

A distance map is simply defined to be a *shortest path tree* (SPT) for some *graph*. Graphs and shortest path trees are explained very briefly below.

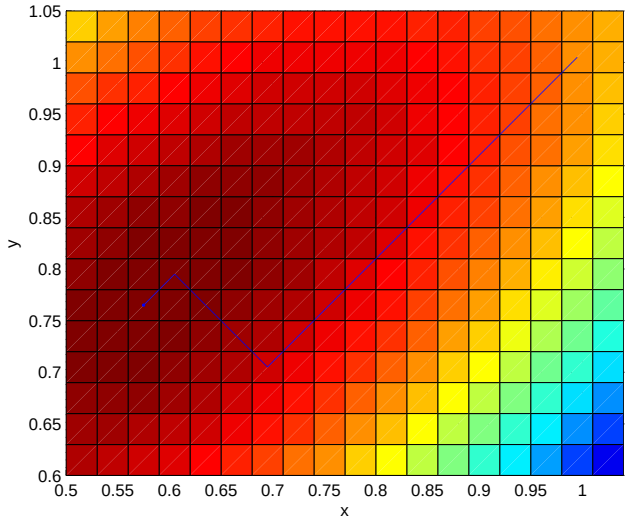
A graph is often written as $G = (V, E)$, where V is a set containing so called *nodes* (or *vertices*), and E is a set of paired nodes referred to as *lines* (or *edges*), describing which nodes that are connected. A simple example of a graph is where nodes denote cities and lines roads between cities. One type of graph is called *weighted graphs*, in which each line is assigned a so-called *weight*. Such a weight represents the cost of moving between the nodes connected by a line. For example, cost might be distance, time, etc.

A SPT constructed out of some graph $G = (V, E)$ solves the so called *single-source shortest path problem* for that same graph G. Basically, it is the problem of finding the shortest paths between a root node $v_0 \in V$ to all other nodes in V. "Finding the shortest path" refers to minimizing the sum of the weights along the constituent lines of a path. A SPT indirectly solves this problem because it annotates all nodes in V with the minimum cost to v_0. Let $d(v)$ denote the minimum cost from an arbitrary $v \in V$ to v_0. It is clear that $d(v_0) = 0$. Nodes that are unreachable from the root node are often annotated with an ∞. A node is unreachable simply due to there are crucial lines missing. For example, consider a town A where no road leads to it, then $d(A) = \infty$ if A is not the root town. Then, finding the shortest path between a $v \in V$ to v_0, where $v \neq v_0 \wedge d(v) \neq \infty$, is a process of repeatedly moving to the neighboring node (as defined by E) with least annotated cost.

It has not been mentioned previously, but G is actually a *hexagon graph*. We defined this type of graph to be a unit distance graph corresponding to the hexagonal lattice. This type of constraint on G is due to a very important property of hexagon graphs;



(a)



(b)

Figure 5.2: Gradient ascent in a discrete version of (5.3) with 8-connectivity neighborhood. The start position was set to equal the cell closest to $(1, 1)$, i.e., the start position used in the example shown in Figure 5.1. (a) shows the result from *discrete descent(ascent) method 1*, and (b) the result from *discrete descent(ascent) method 2*. Both methods are explained in Section 5.1.2.

isotropy. In a broad sense, it means "uniformity in all directions." In this context, it refers to the fact that the neighbors of any node in a hexagon graph are located at an equal distance away. Actually, the RFID tags in PEIS Home have been laid with precisely this property in mind. The advantages by using hexagon graphs are great; it greatly simplifies the off-line calculation of a SPT. More importantly, it makes the on-line calculation of a SPT *possible*, which is because only one neighbor distance must be considered.

More simplified, a distance map can bee seen as a discrete function, with the following definition

$$f : D \to \mathbb{Z}$$

where $D \in \mathbb{N}^2$. Connecting to the graph discussion above; the domain of f is a collection of coordinates that represents nodes in G, and the codomain is a collection of minimum cost values. Unreachable nodes are represented by a negative value, i.e., if $f(\mathbf{x}) < 0$ then the node represented by $\mathbf{x}$ cannot be reached. Let the set $S = \{\mathbf{n}_1, \mathbf{n}_2, ..., \mathbf{n}_6 : \mathbf{n}_1, \mathbf{n}_2, ..., \mathbf{n}_6 \in \mathbb{Z}^2\}$ denote the neighbors of an arbitrary $\mathbf{x} \in D$, expressed in relative coordinates. Note that since f represents a hexagonal grid, it is not possible to define the neighborhood using a single set. It is ignored here to simplify. Some hexagonal coordinate systems even require D to be three-dimensional. Regardless, some characteristics on f can be stated.

- f has exactly one minimum. That is, $\exists!\mathbf{x}^* \in D : \forall(\mathbf{x} \neq \mathbf{x}^*) \in D : f(\mathbf{x}^*) < f(\mathbf{x})$.
- If the minimum of f is $\mathbf{x}^*$, then $f(\mathbf{x}^*) = 0$.
- Let $\mathbf{x} \in D$ represent a reachable node, and $S_v = \{(\mathbf{x} + \mathbf{n}) \in D \wedge f(\mathbf{x} + \mathbf{n}) \geq 0 : \mathbf{n} \in S\}$ be the set of all valid neighbors of $\mathbf{x}$. Then $\forall \mathbf{x}_n \in S_v : |f(\mathbf{x}_n) - f(\mathbf{x})| \leq 1$. This is because each line in G have weight 1.

To find the root node of f from any given start node, it is possible to use the discrete gradient descent(ascent) algorithms described in Section 5.1.2.

5.2.2 Storage of Distance Maps

In the introduction chapter it was mentioned that distance maps are to be stored in the RFID tags located in PEIS Home. For practical reasons this approach was actually not used. Instead an equivalent, more convenient method was used that allows the following.

- A print-out, or visualization of the data attached to each RFID tag at any given time.
- The ability to easily change the data, e.g., insert an off-line constructed distance map.
- The ability to visualize which tags that currently is in the reader's range.

All this was made possible because the following information was available of each tag in PEIS Home: the UID and the location. From an implementation point of view, a virtual RFID tag array was devised and used in the control application, i.e., the Tag Viewer application described in Section 3.2.6. Using this virtual RFID tag array, only the UIDs of the physical RFID tags are required.

An example of visualization of RFID tag data is shown in Figure 3.5, presented by a hexagonal tiling, where each color represents a certain integer value. The shown contents are actually an off-line constructed distance map, where pure green represents 0, i.e., the root node or goal. Increasing red saturation represents increasing integer values. (Pure red represent the highest cost present in the map). Gray represents unreachable tags, i.e., tags which is located under obstacles or is unreachable. Black is broken tags or where tags are not present.

5.3 Executing a Distance Map

This chapter has reached a point where executing of a distance map can be discussed, i.e., gradient descent navigation. The idea behind this navigation method is to use a mobile robot equipped with a RFID tag reader, and then work its way to the goal by using RFID tag data (i.e., a distance map) as guidance. The hardware setup is described in more detailed in Section 3.1.

Note that, as mentioned in the introduction chapter, this navigation method does not require a map of the working environment. It is not even necessary to perform position estimation. These two ingredients are often used to achieve navigation. Other very convenient characteristics of gradient descent navigation are that it is not necessary to input neither the goal position nor the start position. The goal position, in particular, is embedded in the distance map.

The algorithm discussion in this section strongly reminds of the discrete descent(ascent) method 2 described in Section 5.1.2, but applied to a physical search problem. The algorithm consists of the following steps.

1. If the goal is in range, i.e., an RFID tag is in range holding a cost value that equals zero, then stop searching. Otherwise go to Step 2.

2. Determining the direction to the RFID tag (in the vicinity) with lowest cost value. This direction is referred to as $\mathbf{v}_{min}$.

3. Instruct the robot to move along the direction defined by $\mathbf{v}_{min}$. During movement, keep track of and avoid obstacles. Also keep track of the lowest cost value along the path of movement. Go to Step 2 when an increase of the cost value is detected, i.e., when the cost value of the tag currently in range is larger than the lowest recorded cost. Furthermore, go to Step 1 whenever the goal is detected.

The above steps are only a top-level description of the gradient descent navigation algorithm. How the most important parts were solved, i.e., calculation of $\mathbf{v}_{min}$ and obstacle avoidance is explained below together with some hardware considerations.

5.3.1 Hardware Considerations

The success of the described algorithm is dependent on some parameters regarding the RFID hardware. In particular, the following should be considered.

The range RFID tag reading can be performed at. The RFID system should be devised so that only RFID tags in the closest vicinity (i.e., relative the RFID antenna) can be detected. In other words, a passive RFID system should be used. Optimally, at most one RFID should be in the reader's range at any given

time. If the position of the RFID tags form a hexagonal lattice, it can be shown that the reader's range must be lower than $S/2$, where S is the distance between neighboring tags, to guarantee at most one tag in range at the same time. In PEIS Home the distance between tags is 26 centimeters, so $S/2 = 13$ cm. The used RFID hardware was found to exceed this limit; as mentioned in Section 3.1.2, normally one or two tags could be in the reader's range at the same time, and in some cases, up to three tags.

The placement of the RFID antenna on the robot base. The antenna must be placed slightly away from the wheel axis, in the front of the robot. This set-up allows $\mathbf{v}_{min}$ to be calculated.

5.3.2 Determining the Lowest Cost Direction

This step of the gradient descent navigation algorithm refers to determining $\mathbf{v}_{min}$, i.e., the direction to the RFID tag holding lowest cost of tags in the closest vicinity. Connecting to the discussion in Section 5.1 of gradient descent in a real-valued function f; determining $\mathbf{v}_{min}$ is the analogue of determining $-\nabla f$. The calculation of $\mathbf{v}_{min}$ is performed in the following way.

1. Let the robot perform a full rotation around its own axis. As rotation is performed, log when RFID tags enter respectively leave the reader's range. The appropriate property to log is not time, but rather the odometric direction, which is simply the direction of the robot commonly expressed in radians. Also log the cost value of each detected tag.

2. The result from Step 1 describes, for each detected RFID tag, the odometric direction when the tag entered respectively left the reader's range. For detected tag i, denote these directions by $entered_i$ and $lost_i$. By using the rotational direction of the robot in Step 1 together with $entered_i$ and $lost_i$, it is possible to estimate the direction for tag i. This is done by simply calculating the direction in-between $entered_i$ and $lost_i$, where the rotational direction is used to remove an ambiguity in the calculation. (There are two directions that could be defined as in-between $entered_i$ and $lost_i$). Denote the estimated direction by dir_i.

3. From the set of estimated tag directions, the direction for the tag with lowest cost value is picked out and defined as $\mathbf{v}_{min}$. The case when several tags can be attached with the lowest cost value is handled by attaching each direction with a constant length (e.g., 1), resulting in vectors. $\mathbf{v}_{min}$ is then set to equal the direction of the averaged vector.

5.3.3 Avoiding Obstacles

When following the lowest cost direction, it is necessary to avoid any obstacle coming too close to the robot. The obvious risk by not doing this is that the robot could get stuck, or even worse, damage some part of its hardware. Moreover, doing obstacle avoidance is not solely for the consideration of keeping the robot from colliding with objects, but also that it could help in guiding the robot during navigation; performing obstacle avoidance can be seen as a correction of the lowest cost direction, since when obstacle avoidance needs to be done, the lowest cost direction is clearly not correct.

Obstacle avoidance was implemented using the fuzzy controller in the KickOff application. The range-finding sonar sensors located on the robot was used for the purpose of providing the fuzzy controller with distance data. Furthermore, the rules were devised to only make used of the sensors in the front, and those placed on the sides, i.e., the rear sensors are not used. This is due to the assumption that reversing will never occur. Another notable characteristic is that obstacle avoidance is performed solely by rotating the robot; no translation is done, so the robot simply tries face away from nearby obstacles. This approach makes it easier to merge with other behaviors, since translation in the obstacle avoidance behavior must not be considered.

A problem with doing obstacle avoidance with pure fuzzy control is due to the static mapping between the input and output; the same input will always generate the same output, i.e., a fuzzy controller does not have a memory. This is why this method of keeping free from obstacles often is referred to as *reactive obstacle avoidance*. The effect of this is the risk of stabilization; the layout of the obstacles could cause an undesired situation where the resulting rotation is zero, even though there are obstacles nearby. The problem is especially apparent with corners. It occurs when the defuzzification stage has to deal with instructions that are conflicting and equally large. For example, rotate left with speed v, but also rotate right with the same speed. This is indeed contradictory, and resolved by setting the resulting rotational speed to zero. A simple way of solving the problem is by rotating out of danger whenever stabilization is detected. This can be implemented by a state machine, which actually results in a fuzzy controller with a very simple memory. This was the approach used to solve the problem.

Chapter 6

Distance Map Building

In Chapter 5 the concept of gradient descent was explained, and how it can be used to achieve navigation. The discussion in that chapter was based on the premise of the existence of a distance map. This chapter, however, explains how a distance map can be constructed both off-line and on-line.

6.1 Off-line Distance Map Building

Before reading this section, see Section 5.2.1; it contains a definition of distance maps, as well a brief introduction to graphs, and the concept of shortest path tree (SPT).

Off-line distance map building refers to construction of a distance map by the usage of the following.

A map of the working environment. That is, a map describing where obstacles are located, and the extension of each obstacle. The purpose of the map in this context is to describe spaces that are within reach of the robot, so narrow spaces and other inaccessible areas should be handled as well, for example, by representing them as obstacles in the map. The map used for PEIS Home is the bird's-eye view of it shown in Figure 3.2. Inaccessible areas were handled by growing the obstacles by a half robot radius, which can be done by simple methods.

A list of all RFID tags in the environment. That is, a list describing the UID and location of each RFID tag. This list for PEIS Home, as pointed out in a previous chapter, was available.

By using the two ingredients above, it is possible to generate a graph G, that can be used as a basis in the construction of a SPT, i.e., a distance map; since the layout of tags in PEIS Home form a hexagonal lattice, it is clear that the list of tags in PEIS Home can be used to produce a hexagon graph G. The purpose of the map is to decide which nodes in G that are isolated, i.e., have no lines to other nodes. This is easily done by examining the map at those locations where tags are located, as defined by the list of tags.

The big advantage with off-line distance map building is speed; a distance map can be generated quickly, since it is done in memory, i.e., it is not required to perform physical activities of any kind. The drawback is the information requirements; it is necessary to gather a considerable amount of information about the working environment.

6.1.1 Map Building by Breadth-First Search

Under the assumption that a hexagon graph $G = (V, E)$ has been constructed, using the description above, leaves the problem of actually constructing the SPT for G. There are several ways of doing this, depending on the structure of the graph in interest. In this particular case, where G is a unit distance graph, it is possible to construct the SPT by relatively simple means; by exploring the nodes of G as performed in breadth-first search (BFS), which is a graph search algorithm. Given a root node $v_0 \in V$, BFS starts by exploring neighboring nodes of v_0. Then, the neighbors of the closest nodes of v_0 are explored, and so on. That is, BFS recursively explores neighboring nodes. To maintain exploration, BFS makes use of a FIFO (First In, First Out) queue, containing the nodes pending for expansion, i.e., nodes whose neighbors is to be explored. The BFS algorithm, slightly modified to produce the SPT, works as follows.

1. For all nodes v in the graph, do the following assignment $d(v) := \infty$. (A definition of d is given in Section 5.2.1).
2. Push the root node v_0 onto the queue. Moreover, let $d(v_0) := 0$.
3. If the queue is empty, this means that all nodes in the graph have been explored. At this point, stop the exploration.
4. Pop a node off the queue. Denote this node by v_c. Then, create a set S containing the neighbors of v_c that so far are unexamined. Finally, for all $v \in S$, do the following assignment: $d(v) := d(v_c) + 1$.
5. Continue from Step 3.

After termination of the above algorithm, the SPT will be defined by d. It should be mentioned that BFS in this section has been depicted as a way of traversing through a graph − which it basically is. But often BFS is defined as a way of finding a node that fulfills some specific criteria, hence the "search" in its name. But modifications, as above, can produce other interesting results.

A small note: To be able to calculate the SPT for a graph with arbitrary (non-negative) weights, other approaches are necessary; the above algorithm only works with unweighted graphs and graphs where all weights are the same. A well-known algorithm able to handle arbitrary (non-negative) weights are *Dijkstra's algorithm*. As with BFS, it maintains a queue of nodes pending for expansion. The difference is that the queue is sorted with respect to the minimum cost to the root node, i.e., d. The node subjected to pop is always the one with least minimum cost. The result of this is that nodes closer to the root node are explored first.

6.2 On-line Distance Map Building

On-line distance map building refers to construction of a distance map by performing physical actions. This is done by moving the robot around, and while it does, update RFID tag data according to some strategy. The idea behind this approach is not to use any of the information that off-line distance map building requires. That is, no map of the working environment is used, and no information about the placement of the RFID tags. Moreover, the way of defining the goal is simply done by physically placing the robot at the goal, before map building start.

6.2.1 Proposed Building Method

The proposed on-line distance map building method consist of two major parts:

An exploration strategy. For a RFID tag to be given a cost value, it must (at some point during map building) be in range of the RFID reader. Therefore, it is important to carefully devise the strategy the robot moves in accordance with. Particularly, it must be guaranteed that all accessible spaces are visited, so each tag will have a chance to be assigned a cost value. For example, merging translation forward with reactive obstacle avoidance cannot guarantee this.

An update strategy. As the working environment is explored according to the exploration strategy, the data of RFID tags are to be read as they enter the reader's range. Depending on fetched data, an update of tag data might or might not be performed, all depending on the strategy. Clearly, the update strategy should be devised so that RFID tag data will eventually constitute a distance map.

The devised exploration strategy is as follows:

1. Let the robot translate without rotation, i.e., move the robot straight forward. During movement, check for obstacles. When an obstacle is too close to the robot, go to Step 2.
2. Avoid the obstacle solely by rotation; turn the robot until movement forward is allowed. When movement forward is possible, go to Step 3.
3. Randomize an angle θ (e.g., in the interval $[0, \pi/2]$). Perform a partial rotation (θ radians), in the same direction as rotation was done in Step 2. When the partial rotation is completed, go to Step 1.

Basically, the exploration strategy consists of moving straight forward when no obstacles are around. When obstacle ahead, the robot is turned away from the obstacle, plus some additional, random rotation. The element of randomness is necessary in order to guarantee an exploration of all accessible spaces.

The devised update strategy is as follows:

1. Set the data of each RFID tag in the working environment to null, i.e., not holding any value. This is henceforth denoted by $undefined$.
2. Let c denote the distance counter. Reset it by performing the assignment $c := 0$.
3. Construct a set S containing the UID of RFID tags that entered the reader's range. S can easily be derived by comparing RFID tags previously in range (i.e., in range in the previous computation cycle) with tags currently in range.
4. Randomly select one UID in S. Denote the cost that is attached to the selected UID simply by $cost$. Increase the distance counter by one. Then, check if any of the following antecedent part of the rules below evaluates true. In such case, perform the corresponding consequent part.

   IF $cost = undefined \mid cost > c$, THEN $cost := c$

   ELSE IF $cost \geq 0$ & $cost < c$, THEN $c := cost$

5. Continue from Step 3.

The update strategy above basically works by placing out a "trail" in the working environment (i.e., the RFID tags). The trail consists of integer numbers, that tells the distance to the goal. To decide exactly what number to store in the tags as they enter the reader's range, a so-called distance counter is used. It is denoted by c above, and holds an estimation of the minimum cost to goal (i.e., the minimum amount of tags that must be visited in order to get to the goal). (Actually, as can be understood from the algorithm, c holds the cost value for the tag that last entered the reader's range).

Using c, and the cost value (denoted by *cost*) of an entering tag, update of *cost* is done as follows: If *cost* is undefined, then *cost* is set to equal c. The same thing happens when *cost* is larger than c. This is sound since c is assumed to provide a better estimation of the minimum cost to goal. In the case when *cost* is smaller than c, then apparently the cost to goal is smaller than previously believed, so c is updated to reflect this fact.

A non-trivial problem is how to handle the case when multiple tags enter the reader's range at the same time (i.e., when the cardinality[1] of S is larger than 1, i.e., $|S| > 1$). The described update strategy randomly selects one tag of interest, ignoring the others. This approach might or might not cause problems, all depending on the amount of tags that enters the range, and how these tags relate to the tags that were previously in range. For example, it should be clear that the proposed method only can work when two consecutive tags selected for interest are neighbors (i.e., two consecutive selections of a UID in S must correspond to adjacent tags). If this is not the case, this can be seen as a missed increase of the distance counter (i.e., a "skipped" tag), which will lead divergence (when comparing to the ideal distance map). There are several reasons why this could occur. An obvious reason is a too fast exploration speed (or a too slow tag reading speed), which causes problems even when it is guaranteed that $|S| \leq 1$. Another (less intuitive) reason is the amount of tag that can be in range simultaneously; a larger reader range facilitates for a larger value of $|S|$, which might lead to problems. A large reader range can be partially remedied by a slow exploration speed, i.e., moving the robot slow enough.

The most interesting question now remains: Applying the above method, will convergence towards the ideal, off-line generated distance map occur? The answer to this question is given in Chapter 7.

[1] The cardinality of some set S is denoted by $|S|$, and gives the number of elements of S.

Chapter 7

Testing and Evaluation

Testing is of most importance when it comes to estimating the performance of a system, but also to identify possible improvements of its constituent parts. Therefore a full chapter has been dedicated to testing of gradient descent navigation, described in detail in Chapter 5, and on-line distance map building, which is explained in Chapter 6.

The actual tools, i.e hardware and software, used throughout testing and evaluation are explained more closely in Chapter 3.

7.1 Gradient Descent Navigation

Testing of gradient descent navigation has been conducted as intuitively would be expected; by repeatedly executing a distance map under the same conditions. "The same conditions" includes keeping the following parameters constant between different runs: speeds (both translation and rotational speeds), spatial starting position, and starting heading. So, efforts have been made to keep the conditions as similar as possible between runs belonging to the same test. Moreover, parameters between different tests have been kept same to the largest extent possible (e.g., speeds), so that comparison between tests can be conducted.

Furthermore, testing has been restricted to off-line generated distance maps only. The most prominent reason for this is to remove the element of error in the distance map; full focus can be kept on the actual gradient descent navigation algorithm, and the distance map can be ruled out as the cause of any problem occurring during testing, since off-line generated distance maps are assumed to be ideal. But of course, it would be interesting to perform gradient descent using a non-ideal, on-line generated distance map. This could be an interesting future work to perform.

7.1.1 Data Collection Strategy

When doing any king of testing, it must be decided precisely what parameters to record, or log. In case of gradient descent navigation (or any other navigation method for that matter), it is obvious that (at a minimum) time and distance should be involved. During all tests, the following was logged.

Total navigation time. That is, the total time elapsed from that the robot is instructed to start navigation, until it finds the RFID goal tag.

Total distance. That is, the length of the path the robot moves according to, in order to get from the starting position to the RFID goal tag. The most intuitive way of achieving this is to make use of the odometric position; if $\mathbf{p}_k$ and $\mathbf{p}_{k+1}$ are two consecutive estimations of the robot's position, then $\sqrt{\mathbf{p}_k \bullet \mathbf{p}_{k+1}}$ (i.e., Euclidean distance) provides an estimation of covered distance between $\mathbf{p}_k$ and $\mathbf{p}_{k+1}$. The total distance is given by repeating the calculation for each two consecutive estimations, and sum up the result. Another way of estimating the position of the robot is to use the known positions of the RFID tags; multiple tags in range at the same time are handled by averaging of their coordinate vectors. Both methods of estimating the position of the robot was used.

Number of rotations. That is, the number of corrections of the lowest cost direction. As described in Chapter 5, a correction occurs when an increase of the cost value is detected. Note that, in the test results below, the initial rotation is included in the figure.

The path. Logging the path of the robot as navigation is performed is very useful for later visualization. Logging was done by periodically storing the estimation of the robot's position; the odometric position was used, but also an estimation using the known positions of RFID tags in the reader's range. Actually, the position estimations calculated in **Total distance** was used directly for this.

A possibly remaining question is as follows: Why is not the position of the robot estimated solely using odometry? The reason is that a robot pose estimated using only odometry will inherently accumulate errors that eventually get out of hand − no matter what action is taken to reduce the errors. Position estimation using the RFID tags known positions does not suffer from this, but the accuracy is not very great; the fairly large distance (26 cm) between the tags in PEIS Home limits the resolution. However, using both approaches is useful since they complement each other; odometry is more accurate in the short run, while RFID tag positions are more accurate in the long run.

During all tests, the translation speed was limited to 80 mm/sec, and the rotational speed limited to 0.4 rad/sec.

7.1.2 Test 1 − Simple Path In Both Directions

This test consisted of running the same path in both directions. This was done by generating and using two distance maps; the starting position of the robot for one map was the same as the goal for the other map, and reversed. This test was fairly easy (with respect to the distance between start and goal); navigation took place between the kitchen and the open space in the living room. Figure A.1 shows the actual goal tags that were used. The tags in question are represented by two circles; one is filled black, the other one is open.

From The Kitchen To The Living Room

One of the sub-tests of this test consisted of navigating from the kitchen to the living room. A total of five navigation runs was performed under as similar conditions as possible. Table 7.1 (a) shows the result of each navigation run.

As seen in Table 7.1 (a), the stability of the navigation runs is fairly stable, both with respect to covered distance and navigation time; the measured distances differ at most a few decimeters from the average values. This applies both for position estimation

using odometry and the RFID tags. Position estimation using the RFID tags, however, appears to give higher distance estimations.

The most notable test runs in Table 7.1 (a) are 2 and 5, since they are the two extremes with respect to navigation time. Test run 1 managed the navigation in only 94 seconds, while test run 5 needed 127 seconds. The long run time for the latter is because three rotations were required. Figure A.1 shows the recorded paths for the test runs in question. Figure A.1 (a) and Figure A.1 (b) shows the paths using odometry and tags for position estimation, respectively. In both sub-figures, a filled circle represents the goal tag, and an open circle approximately represents the starting position. Moreover, a solid line is the path of test run 2, while a dashed line is test run 5.

As seen in Figure A.1 (a), the odometry is far from reliable; especially look at where the path ends for test run 5. The reason why paths generated out of odometry is shown in this report is due to the smoothness; it is considerably easier to deduce how the robot actually moved.

Using Figure A.1, it is possible to determine the reason behind that extra rotation required for test run 5. As can be seen, the robot missed the goal, which later led to an extra rotation due to the detection of an increasing cost value.

From The Living Room To The Kitchen

The second sub-test consisted of navigating from the living room to the kitchen. As in the previous sub-test, a total of five navigation runs was performed. Table 7.1 (b) shows the result of each navigation run.

As with the previous sub-test, the stability of the navigation runs is fairly stable, both when considering covered distance and navigation time. As seen in Table 7.1 (b), all measure parameters are generally lower compared to navigating in the opposite direction.

Two interesting test runs (if time is considered again) in Table 7.1 (b) are 2 and 4. Test run 2 managed the navigation in 96 seconds, while test run 4 only needed 67 seconds. As before, the time difference is due to difference in the number of rotations. Figure A.2 shows the recorded paths for the test runs in question. Figure A.2 (a) and Figure A.2 (b) shows the paths using odometry and tags for position estimation, respectively. In both sub-figures, a solid line is the path of test run 4, while a dashed line is test run 2.

The cause of the extra rotation in test run 2 is because the initial calculated lowest cost direction differs. This is easiest to see in Figure A.2 (a).

7.1.3 Test 2 – Difficult Path In Both Directions

This test was performed exactly in the same way as Test 1, which is described in Section 7.1.2. The difference is that a considerably tougher path was executed. The intention was to see how a more difficult path handles by gradient descent navigation. The goal tags of the actual paths was located in the kitchen and the bedroom, which Figure A.3 depicts. The tags in question are represented by two circles; the circle filled with black is the kitchen tag, and the open circle is the bedroom tag.

The difficulty of the path is not only due to the distance, but also that the robot is required to navigate through the relatively narrow opening that connects the bedroom and the living room. Below is the result of navigation from the kitchen to the bedroom, and in the opposite way.

	Distance (m)			
Test Run	Odometry	RFID Tags	Navigation Time (s)	Rotations
1	4.71	4.85	112	2
2	3.69	4.32	94	2
3	4.32	4.58	110	2
4	4.54	5.14	108	2
5	4.21	4.34	127	3
Min	3.69	4.32	94	2
Average	4.30	4.65	110	2.2
Max	4.71	5.14	127	3

(a) Logged data from navigation between the kitchen and the living room.

	Distance (m)			
Test Run	Odometry	RFID Tags	Navigation Time (s)	Rotations
1	4.21	4.62	86	1
2	3.27	3.51	96	2
3	3.01	3.19	69	1
4	2.89	3.41	67	1
5	3.14	3.72	95	2
Min	2.89	3.19	67	1
Average	3.32	3.69	83	1.4
Max	4.21	4.62	96	2

(b) Logged data from navigation between the living room and the kitchen.

Table 7.1: Data that was gathered during the tests described in Section 7.1.2, i.e., navigation between the kitchen and the living room, but also in the opposite direction.

From The Kitchen To The Bedroom

This sub-test consisted of navigating from the kitchen to the bedroom. A total of five navigation runs was performed. Table 7.2 (a) shows the result of each navigation run.

Looking at Table 7.2 (a), the test runs show the similar result when it comes to covered distance, i.e., about 10 meters. However, an exception is present; test run 4 only required circa 7.5 meters to find its way to goal. Moreover, test run 4 has the shortest navigation time. Test run 3 is another exception, because it has the longest navigation time, which is due to the large amount of rotations it required.

Since test run 3 and 4 shows some interesting exceptions, there recorded paths are shown in Figure A.3; Figure A.3 (a) and Figure A.3 (b) shows the paths using odometry and tags for position estimation, respectively. In both sub-figures, a solid line is the path of test run 4, while a dashed line is test run 3.

During test run 4, as can be seen in Figure A.3, the robot managed to stay close to obstacles, which is why the resulting was path so short. In test run 3, the cause of the many rotations was because the estimation of the lowest cost direction was off. This is particularly easy to see in Figure A.3 (a); the robot zigzagged its final meters.

From The Bedroom To The Kitchen

This sub-test consisted of navigating from the bedroom to the kitchen. A total of five navigation runs was performed. Table 7.2 (b) shows the result of each navigation run.

As seen in Table 7.2 (b), the result shows good stability; no big deviation is present in any of the measured parameters. This could be explained by luck. Moreover, the measured distances and times are generally lower than in the previous sub-test, but not by very much.

To conclude this section, test runs 1 and 2 (the worst and the best runs with respect to navigation time) are shown in Figure A.4; Figure A.4 (a) and Figure A.4 (b) shows the paths using odometry and tags for position estimation, respectively. In both sub-figures, a solid line is the path of test run 1, while a dashed line is test run 2.

7.1.4 Test 3 – Moderately Difficult Path In One Direction

In Test 1 and Test 2, described in Section 7.1.2 and Section 7.1.3, respectively, there was only a few test runs made in each direction. As interesting as these tests might be, statistical calculations of the result are not feasible, simply due to the few runs. Therefore, it was decided to perform a larger amount of test runs; a total of 50 test runs was performed from the kitchen to the lower parts of the living room, which is a path of average difficulty. Figure A.5 (a) shows the location of start and goal; the filled circle is the goal, while the open circle is the start.

Table 7.3 shows some statistics calculated out of the 50 test runs. Here follow a few comments regarding the table:

Distance: Looking at the calculated confidence intervals, it is clear that a large majority of the measured distances fell into a relatively narrow interval; for both estimation methods, this interval is circa 0.3 meters, which should be considered as good. However, as seen in the table, there were at least one test run with a covered distance as low as 4.3 meters, and at least one as high as 6 meters. Having the confidence intervals in mind, these are exceptions. The estimated standard deviations are another indicator of the similarity between the test runs; it is close to 0.35 meters in both position estimation methods.

	Distance (m)			
Test Run	Odometry	RFID Tags	Navigation Time (s)	Rotations
1	9.01	9.20	198	3
2	10.62	10.22	237	4
3	9.90	9.83	284	6
4	7.31	7.46	168	3
5	9.25	9.95	229	4
Min	7.31	7.46	168	3
Average	9.22	9.33	223	4
Max	10.62	10.22	284	6

(a) Logged data from navigation between the kitchen and the bedroom.

	Distance (m)			
Test Run	Odometry	RFID Tags	Navigation Time (s)	Rotations
1	9.10	9.28	239	5
2	7.59	7.87	175	3
3	7.86	8.21	205	4
4	7.38	7.61	191	4
5	8.65	8.72	236	5
Min	7.38	7.61	175	3
Average	8.12	8.34	209	4.2
Max	9.10	9.28	239	5

(b) Logged data from navigation between the bedroom and the kitchen.

Table 7.2: Data that was gathered during the tests described in Section 7.1.3, i.e., navigation between the kitchen and the bedroom, in the opposite direction.

Navigation time: According to the confidence interval, the navigation time of most runs fell into between 121 and 140; a corridor of 19 seconds in not bad at all. As usual, there are exceptions; one run managed to find the goal in only 87 seconds, while another run needed as much as 180 seconds. Important to note is that the navigation time calculations are not fully truthful; navigation time is directly dependent on the number of rotations. Still, the calculations are interesting because they provide an average over all runs.

Rotations: The calculations indicates that usually two or three rotations were required. This agrees with what one might intuitively expect; the initial rotation, another rotation when going into the open space in the living room, and on some cases, a third rotation occurring when the goal is missed. The extreme cases are one and four rotations.

	Distance (m)			
Property	Odometry	RFID Tags	Navigation Time (s)	Rotations
Min	4.31	4.35	87	1
Median	5.21	5.38	128	2
Average	5.20	5.35	130	2.5
Max	5.99	6.07	180	4
Sample SD*	0.37	0.36	24.24	0.96
99% CI**	(5.05, 5.34)	(5.21, 5.50)	(121, 140)	(2.13, 2.87)

* SD = Standard Deviation
** CI = Confidence Interval

Table 7.3: The above table shows statistics that was calculated out of gathered data from the test described in Section 7.1.4, i.e., navigation between the kitchen and the living room.

To conclude this section, two extreme runs (with respect to rotations) are shown. The test runs in question are number 3 and 25. There recorded paths are shown in Figure A.5; Figure A.5 (a) and Figure A.5 (b) shows the paths using odometry and tags for position estimation, respectively. In both sub-figures, a solid line is the path of test run 25, while a dashed line is test run 3. The logged data for the test runs are shown in Table 7.4.

According to Table 7.4, the covered distance of test run 3 and 25 is similar, yet the number of rotations differs greatly. The answer lies in Figure A.5 (a), and the cause is interesting; in test run 25, the robot approached the second obstacle (the chairs, actually) more directly, resulting in a more aggressive action by the obstacle avoidance routine, leading to a more direct route to goal. In test run 3 however, the robot approached the chairs less directly, resulting in a weaker action by the obstacle avoidance routine, leading to a less direct route to the goal and more rotations.

7.1.5 Conclusions

In this section has gradient descent navigation been applied to several problems. The overall impression is more than positive − considering that very little information is used during navigation. Regarding stability, i.e., the similarity between measured parameters

	Distance (m)			
Test Run	Odometry	RFID Tags	Navigation Time (s)	Rotations
3	5.24	5.79	168	4
25	4.76	5.11	87	1

Table 7.4: Data that was gathered for two of the most extreme runs during the test described in Section 7.1.4 − extreme with respect to the number of rotations that were required. The test consisted of navigation between the kitchen and the living room.

of repeated runs, is also positive. However, as has been indicated several times in the performed tests, it happens that a test run differs quit much from the average run. This is likely explained by the fact it is hard to maintain the exact starting pose between test runs; one or two centimeters in the position, and a few degrees in heading are not unlikely. As discussed at the end of Section 7.1.4, a seemingly insignificant difference can become apparent later on.

7.2 Distance Map Building

In Section 6.2, it was proposed how on-line distance map building can be conducted. In that section however, it was not indicated of proved that convergence actually occurs, i.e., to the ideal, off-line generated distance map. This section attempts to prove the validity of the algorithm by systematic evaluation. That is, by running tests.

Intuitively, a good way of conducting testing is by performing map building several times, using the same root node, or starting RFID tag, in each build. This could be repeated for several root nodes. Unfortunately, due to time limitations, this was not realistic. Instead, it was decided to do two map builds that together (to some degree) prove the validity:

(1) Building in a small space, until convergence. That is, map building in a partial space of PEIS Home, up until the point convergence occurs, or when any other final conclusion can be drawn.

(2) Building in the full space, but not all the way to convergence. That is, map building using all available space in PEIS Home, up until the point where a preliminary conclusion can be stated.

If convergence is obtained in (1), it will prove that the proposed method works (at least on a small scale). If the outcome from (2) is good, it will provide an indication that it also works on a larger scale.

In all tests described in this section, the translation speed was limited to 40 mm/sec, and the rotational speed limited to 0.4 rad/sec. These speeds are actually close to the lowest possible of the robot that was used (lower speeds, for various reasons, result in rounding to zero). The reason why these low speeds were chosen was to minimize the risk for the problem described at the end of Section 6.2.1. That is, tags that are given a lower associated cost than intended. As mentioned in Section 3.1.2, the actual RFID reader that was used, together with the layout of the RFID tags in PEIS Home, allowed up to three tags to be in range simultaneously. Furthermore, it was found by testing that, when the robot performed translation at 40 mm/sec at most two tags could enter

the range at the same time. The mentioned problem should not be present in this set-up, which is confirmed by the test results below.

7.2.1 Difference Between Distance Maps

In order to visualize (i.e., plot) the difference between an ideal distance map, and an on-line constructed one versus time, it is a necessity developed some sort of error measurement. A measurement has been devised that reminds of the well-known Euclidean distance for the Euclidean space: The error between two arbitrary distance maps G_1 and G_2 (defined over the same graph $G = (V, E)$) is as follows.

$$dist(G_1, G_2) = \sqrt{\sum_{v \in V} (d_v)^2} \tag{7.1}$$

In (7.1) is d_v defined as $d_v = d_1(v) - d_2(v)$, where $d_1(v)$ and $d_2(v)$ refers to the associated cost of v, defined by G_1 and G_2, respectively. (A more explanatory description of d is given in Section 5.2.1). Note that (7.1) is not directly usable; it is necessary to define the cases when $d_1(v) = \infty$ or $d_2(v) = \infty$, or both, else d_v is clearly not defined. This is handled by replacing d_v with some value according to Table 7.5. As seen in the table, when exactly one $d_1(v)$ and $d_2(v)$ equal ∞, then replacement is done using a constant M. This constant represents the maximum possible cost associated with any $v \in V$, with G in mind (i.e., the working environment). In other words, M basically represents infinitive, but in "practice" so that evaluation of (7.1) is made possible. When building in the full space of PEIS Home, a reasonable value for M is 50; it is unlikely that this value would be exceeded even during an on-line build.

This approach of dealing with undefined associated costs is good, especially during comparison of an ideal and an on-line built distance map; (1) tags that has not yet been visited (but should be, according to the ideal map) can be said to be "penalized"; (2) unvisited tags located farther away from the goal is penalized less than unvisited tags located closer to the goal. (1) is good because more visited tags gives a more informed map. (2) is coherent with the fact that it is more important to explore tags that are closer to the goal; the associated cost of a tag closer to the goal will affect the cost of many more tags than a tag located far away from the goal.

It should be emphasized that if the above error measurement is used for comparison of an off-line and an on-line built distance map, then actually two pieces of information are merged; (1) the extent of the exploration (i.e., the number of tags visited); (2) the quality of the cost values of visited tags. (1) is due to the penalization of unvisited tags, and (2) is due to the contribution of d_v. Note that, since the penalization of tags is likely to dominate in (7.1) (especially in the earlier stages of a build), there is the risk that the contribution of (2) is barely detectable. This is a problem when plotting the error versus time. A solution (and the solution that was used) is to use a logarithmic scale for the error axis. This will help to show the nuances of (2), particularly when most tags have been visited (when the major contribution to (7.1) not is penalty terms). Another solution (that was also used) is to decouple (1) and (2), and make use of two plots; a plot for showing the amount of tags left to visit (or the amount of tags visited), and a plot of (7.1) but where the penalty is zero.

	$d_1(v) = \infty$	$d_1(v) \neq \infty$
$d_0(v) = \infty$	0	$M - d_1(v)$
$d_0(v) \neq \infty$	$M - d_0(v)$	−

Table 7.5: This table shows replacement strategies for d_v in (7.1), to handle those cases when it is undefined, due to infinitive values associated with $d_0(v)$ or $d_1(v)$. The "−" means that no replacement action is to be performed.

7.2.2 Expectations

Although no test result has been presented yet, it is not difficult to imagine possible outcomes. Given an ongoing on-line build, whose distance map is denoted by G, and an ideal, off-line generated distance map for the same root node, denoted by G^*, then one of the following (with respect to $dist(G, G^*)$) should occur after a long enough build.

Converge to zero. That is, given enough time, the on-line constructed distance map eventually equals the ideal one, i.e., $dist(G, G^*) = 0$, and stays there. This is the most desirable case of all.

Converge to a constant steady-state error. That is, $dist(G, G^*)$ will eventually equal some constant c, and hold this value for an infinite time. This result occurs when G^* is not actually ideal; perhaps the map of the working environment used in the construction of G^* is outdated or incorrect. Another explanation is the presence of broken RFID tags; a broken RFID tag cannot be detected, and will consequently not trigger an increase of the distance counter.

Oscillate around some limit value. Here the idea is that, given enough building time, oscillation occurs, and continues to do so for infinity. It is difficult to reason about the cause of such behavior. Likely tough, the update strategy is faulty.

Diverge. This means that, as building elapses, the difference between G and G^* increases. This is the most serious outcome of all, since it suggests on-line distance map building is not possible. Assuming that the update strategy is valid, then a likely explanation of divergence is the problem described at the end of Section 6.2, which (as mentioned in Section 6.2) can be due to "skipped" RFID tags, i.e., the robot is moving too fast compared to the time needed for RFID tag reading, leading to missed increases of the distance counter.

The best possible outcome from the following tests is not convergence to zero, as might be believed, but rather a convergence to a constant steady-state error; there are several broken RFID tags in PEIS Home.

7.2.3 Test 1 – Building in a small space

In this test, the robot was enclosed inside the bedroom of PEIS Home, simply by blocking the way out of there. Figure 3.1 (a) shows a photo taken just outside the bedroom entrance. Before building started, the robot was placed in the middle of the room, thus defining the goal RFID tag (i.e., the root node).

In this test, as mentioned in the introduction of this section, the intentions were to perform building until such time a definite conclusion could be drawn. Actually, it was

found that the build converged to a constant steady-state error, and that the final distance map looked very similar to the ideal one. This is a very good result. Furthermore, to confirm this was actually case, see Figure 7.1 and Figure 7.2. Figure 7.1 shows the error in the built map versus building time, as defined by the error measurement (7.1). Figure 7.2 (a) shows the resulting distance map (the screenshot is from the control application described in Section 3.2.6). As seen in Figure 7.1, convergence occurred after about 5400 seconds, at an error of 3. Figure 7.2 (a) reveals why this is so; it is because of the presence of broken tags. Note in particular the broken tag located above the goal. Compare to the ideal map is shown in Figure 7.2 (b). Also note that the build was allowed to continue all the way to 6600 seconds − this to provide a better indication that convergence actually occurred.

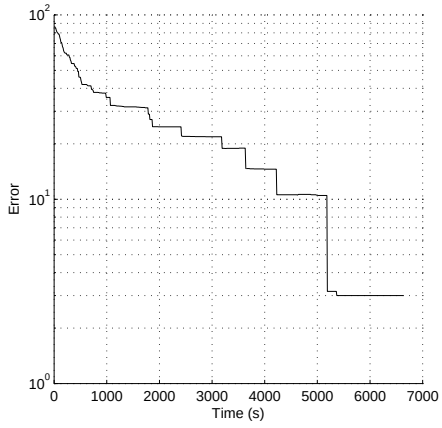
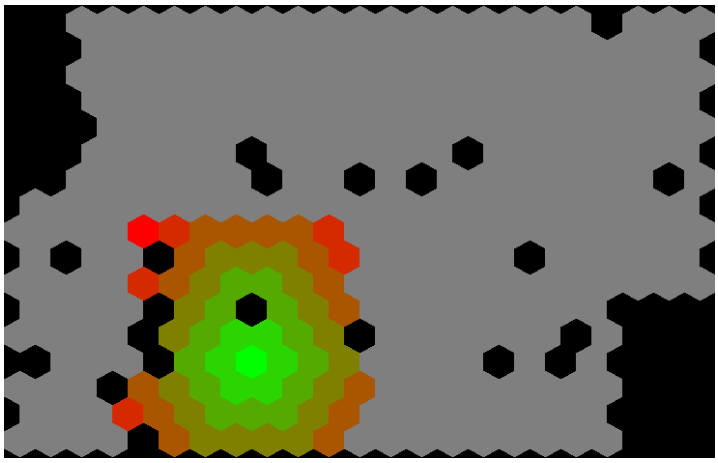


Figure 7.1: A plot from the on-line distance map build described in Section 7.2.3. The plot shows the error in the map build (compared to the ideal) defined by (7.1), with $M = 15$. Only unvisited tags (that should be visited according to the ideal map) has been penalized. Note that the error axis uses a logarithmic scale. See Figure 7.2 for an image of the map when the build was interrupted, and the ideal map.

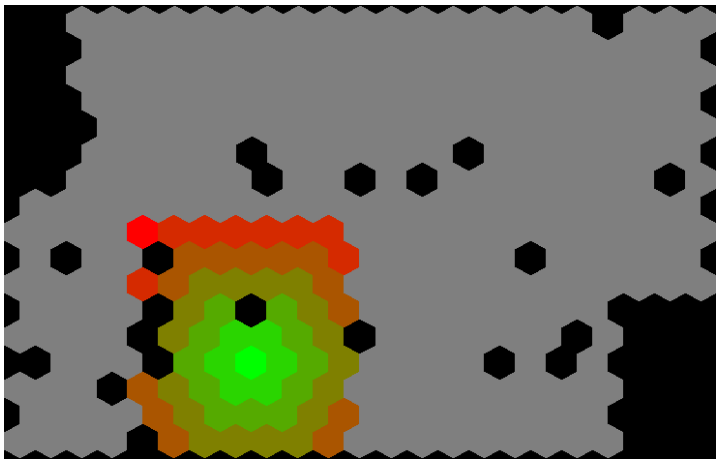
7.2.4 Test 2 − Building in the full space

In this test, a map build was performed using the full space in PEIS Home. As mentioned in the introduction of this section, the purpose was not to perform building all the way convergence, but enough to satisfy a preliminary conclusion. The root node for this test was located in the living room. More precisely, located between the shelf and the TV bench shown in Figure 3.1 (c).

Total build time for this test was six hours. The collected data suggests that convergence would happen if building continued, and the visualization of the resulting distance map looks very promising. Figure 7.4 contains a few plots from this experiment; Figure 7.4 (a) shows the error in the built map versus building time, as defined by the error measurement (7.1). Note that the error axis this time is linear; the build did not reach a point where a logarithmic axis could be motivated. Figure 7.4 (b) and (c) constitute the decoupled version of Figure 7.4 (a); Figure 7.4 (b) shows the total error of the visited tags only (i.e., no penalty terms have been added); Figure 7.4 (c) shows the (relative)



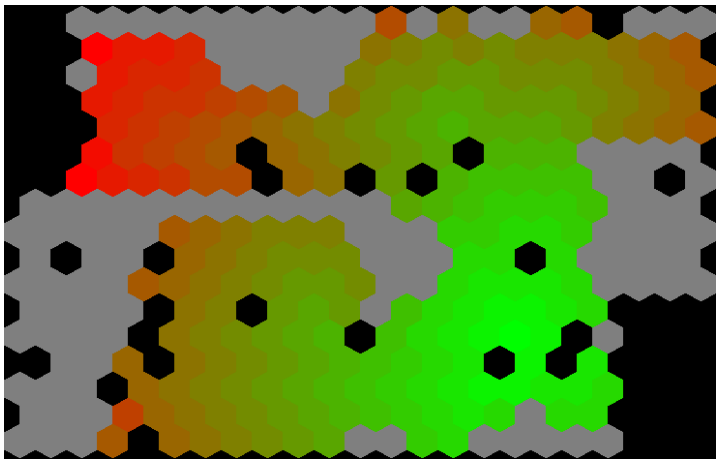
(a) Resulting distance map



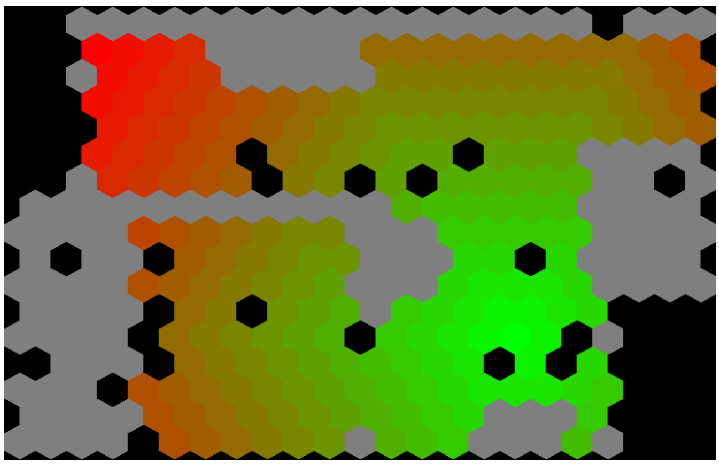
(b) Ideal distance map

Figure 7.2: Two images that resulted from the on-line distance map build described in Section 7.2.3: (a) shows the resulting distance map. (b) shows the ideal distance map. See Section 5.2.2 for a description of the meaning of all colors.

amount of visited tags versus time. See Section 7.2.1 for a discussion about the reasons behind decoupling. In Figure 7.4 (b), the very rapid increase at $t = 1000$ is because the robot entered the kitchen with a large value of its distance counter. At about $t = 6200$, the robot managed to enter the kitchen again, but this time with a lower (and more correct) distance counter. Looking at Figure 7.4 (c), it is seen that almost all RFID tags were visited at least once; this should be considered as very good, and suggests that the exploration strategy works. The overall trend of the plots suggests that convergence will occur; all three plots appear to closing in on their optimal values. Figure 7.3 (a) shows the resulting distance map. As seen, it reminds very much of the ideal, off-line generated distance map shown in Figure 7.3 (b). A few more hours of building and it will most likely converge (to a constant steady-state error, since several RFID tags are broken).

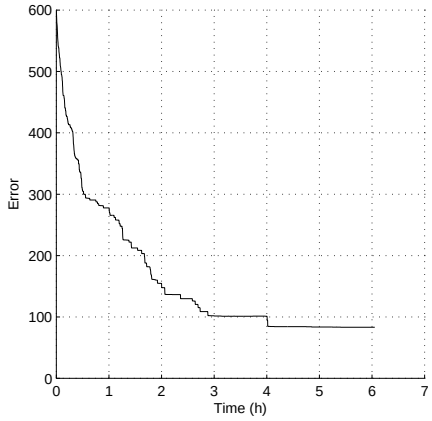


(a) Resulting distance map

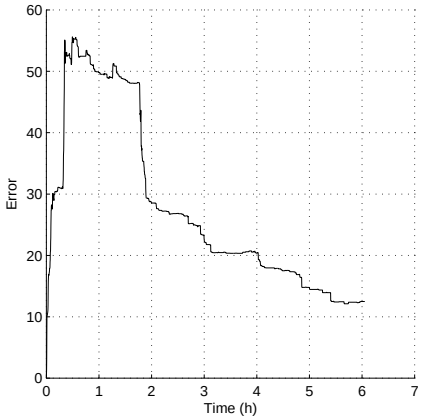


(b) Ideal distance map

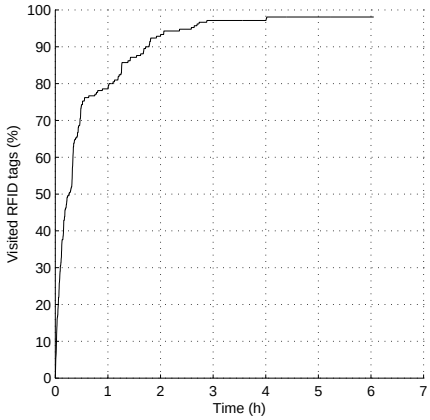
Figure 7.3: Two images that resulted from the on-line distance map build described in Section 7.2.4: (a) shows the resulting distance map. (b) shows the ideal distance map. See Section 5.2.2 for a description of the meaning of all colors.



(a) Estimated error (with penalty) (linear error axis)



(b) Estimated error (no penalty)



(c) Visited RFID tags in percent

Figure 7.4: Resulting plots from the on-line distance map build described in Section 7.2.4: (a) shows the error in the map build (compared to the ideal) defined by (7.1), with $M = 50$. Only unvisited tags (that should be visited according to the ideal map) has been penalized. (b) shows the same as (a), but no penalties has been applied at all. (c) shows the number of visited tags (i.e., tags that should be visited according to the ideal map).

7.2.5 Conclusions

Although no formal proof has been presented, the result in this section strongly indicates that on-line map building actually works; it has basically been proved (by building until convergence) that it works on a smaller scale, and the result suggests it to also be working on a larger scale.

As has been indicated in the tests, broken RFID tags prevents convergence all the way to the ideal map. Still, even if there are broken tags present, convergence will occur to a map that reminds much of the ideal one. That is, convergence to a constant steady-state error.

Chapter 8

Discussion and Conclusions

This chapter gives an overview of what has been done, conclusions, and future work proposals.

8.1 Work Overview

The main objectives of this project were to propose a method of doing indoor robot navigation influenced by stigmergy. The term stigmergy refers to the communication that takes places between insects through modification of the environment, and allows different types of optimization problems to be solved with no or little *a priori* information. The proposed navigation method is "influenced" by stigmergy in the sense that very little information is required to manage navigation. Actually, the navigation method proposed in this report does not need any of the following: a map of the working environment, position estimation, input the starting position of the robot, and defining the goal. The only thing it relies on is "trails" placed out in the working environment. By letting the robot follow such a trail, it finds its way to the goal. Ants place out trails in the environment by the means of pheromone − a chemical substance. For practical reasons, pheromone (or any other chemical substances) has not been used. Instead, so called RFID tags have been used for the purpose of modifying the environment. RFID tags can be seen as (actually, they are) a general-purpose, wireless-accessed storage device. This means that a navigation robot was equipped with a *RFID tag reader* − a device capable of fetching and modifying the data stored in a RFID tag. The actual "trails" are in this report referred to as a *distance map*. See Section 5.2.1 for a closer definition. Basically, when the information stored in RFID tags constitute a distance map, then each tag holds an integer value that tells the minimum amount of tags that must be passed in order to get to the goal. Using these associated integer values, it is possible to achieve navigation, which is referred to as *gradient descent navigation*. As might be known, gradient descent is a *gradient informed* optimization algorithm intended for multi-variable functions. The proposed navigation method reminds of gradient descent, hence the name.

Gradient descent navigation requires a distance map to work, therefore work has been spent on proposing an algorithm that allows construction of such a map by an *online* approach. This refers to the construction of a distance map by not using any prior information about the working environment *at all*; the robot performs an exploration of the working environment (i.e., drives around), and at the same time, updates the as-

sociated integer values according to a certain strategy. In this approach, distance maps are *converged*; the longer building time, the better the distance map becomes. Furthermore, it has been shown how distance maps are constructed by an *off-line* approach. In this method, no physical actions are performed, but calculated by an application by the use of a common graph search algorithm. Building by the off-line method requires a considerable amount of information about the working environment, which is to a great disadvantage because the stigmergic touch (i.e., low information requirements) is lost.

8.2 Conclusions

To briefly conclude, the main contributions are

- It is proposed how modification of the environment can be achieved by the use of RFID tags – for the purpose of achieving navigation. Specifically, a concept referred to as *distance maps* has been devised, which defines how tags should be positioned in the working environment, and what each tag should store in order to maintain navigation.

- It is proposed how a distance map can be constructed in both *on-line* and *off-line* manners. In the on-line approach, it is possible to construct a distance map with no *a priori* information at all, by physically moving a robot around in the working environment. This clearly agrees with the low information requirements stated in the original goals of the project (see Section 1.3.2). In the off-line approach, are distance maps calculated by an application. This approach requires a considerable amount of a priori information, but the construction is very fast. Including a proposal of how to do off-line building was not a part of the original goals of the project, but was included anyway, for completeness.

- It is proposed how a distance map can be executed by a mobile robot, i.e., proposed how a distance map can be utilized in order to maintain navigation. The proposed navigation method is called *gradient descent navigation*, and differs considerably compared to more classical ideas of achieving navigation. Most prominently, the following is not required at all: a map of the working environment, defining the starting pose of the robot, position estimation, and state the location of the goal. The latter might appear at bit odd, but is not required since the goal is implicitly defined by the distance map. To conclude, navigation by the proposed method requires no a priori information (except for the distance map), which agrees completely with the project goals (see Section 1.3.2).

- It is important to emphasize that all constituent parts of the proposed navigation method can be achieved using no a priori information; a distance map can be built by no a priori information, and gradient descent navigation only requires (in advanced) a distance map to work.

Considering the tests that have been performed, the proposed navigation and map buildings methods show some very promising results. In particular, the proposed on-line map building method has basically been proved to be working by empirical, systematic evaluation. Furthermore, gradient descent navigation has also been evaluated by a systematic approach, and considering that it managed some rather difficult paths, we can with a good assurance state that this also works.

8.3 Future work

The following is a few proposals for future work, that directly continues on the work presented in this report.

Store distance maps in the RFID tags. Throughout the work during this project, distance maps were never actually stored physically in the RFID tags. Storage was instead realized by implementing and using a virtual RFID tag array. That is, the purpose of the RFID tag reader was only the fetch the UIDs of tags in the reader's range; the associated cost was fetched by indexing the virtual array by an UID. A possible future work is to skip the virtual array, and try to use the tags for storing of a distance map.

Why was this not done? There are several reasons why this approach was selected, but mainly out convenience and for debugging purposes. See Section 5.2.2 for more details.

Difficulty? Easy, since it only would be required to do some relatively simple changes in the control application described in Section 3.2.6.

Map building with multiple robots. This is, try to do on-line distance map building by the method described in this report, but with two or more robots. Since multiple units are required to have access to the same distance map, it must be shared between them. The intuitive solution is to store distance maps in the RFID tags them self. This approach, however, lacks the advantages of a virtual RFID tag array. A better approach is to use a *shared virtual RFID tag array*. That is, the associated costs of the tags are managed by some central unit, that is accessible by each robot. When it comes to actual map building with multiple robots, interesting questions include: will convergence occur? will convergence happen faster? will n robots result in convergence n times as fast?

Why was this not done? There was simply only one robot available. Even in the case of an extra robot, it would have been difficult to find time.

Difficulty? Medium, since it would be required to somehow share the distance map between the robots. PEIS Kernel could help in achieving this.

Map building in full space. Due to time limitations, it was not possible to evaluate the propose on-line distance map building algorithm as much as desired. It was shown to be working on a smaller scale. More tests should be performed using the full space in PEIS Home. For example, it would be interesting to perform map building until convergence, or to any other conclusive result occur, multiple times with the same root note (goal), and compare the individual builds with respect to some parameters (e.g., building time).

Why was this not done? There was simply not enough time to be able to perform this kind of exhaustive tests.

Difficulty? Medium, since it would be time consuming to perform really exhaustive tests.

Gradient descent with a converged distance map. In this project, we satisfied with only performing gradient descent on off-line constructed distance maps. A future work could be to evaluate gradient descent on a converged on-line constructed

distance map. That is, converged in the sense that a constant steady-state error has been achieved; the broken tags in PEIS prevent a full convergence. So, the interesting is to see if (and how) this affects gradient descent.

Why was this not done? The reason why gradient descent was only tested on ideal maps, as mentioned in Section 7.1, was to be able to rule out the map as an error source if any problem occurred during testing.

Difficulty? Easy, under the assumption that converged, on-line constructed distance maps exists, it is a simple as executing an off-line correspondence.

Gradient descent at intervals with a non-converged distance map. Another interesting type of test to perform is to "combine" gradient descent and on-line distance map building; at given intervals during a map build (e.g., with respect to build time or degree of convergence), gradient descent is applied. Intuitively, as the degree of convergence increases, the quality of the navigation runs should increase (e.g., with respect to navigation time, the number of rotations, etc.). Another possibility is that navigation will fail (i.e., never reach the goal) until a certain degree of convergence occurs.

Why was this not done? This is a time consuming task, which is why is was not performed.

Difficulty? Medium, mainly because this test requires a considerable amount of time. This is particular the case if the test is to be repeated for different root nodes.

Prove the validity of on-line map building. In this report, the proposed on-line distance map building method has not been formally proved. Instead, its validity has been proved to a certain degree by empirical, systematic evaluation. That is, simply by running tests, and deriving conclusions from the test results. A future work could be to formally prove that the proposed method actually works, by using some appropriate mathematical tool. A first step in proving this could be to assume that at most one RFID tag can be within the range of the reader at any given time − this would probably simplify a proof considerably. Second step is to move on to the more difficult case when several tags can be in range simultaneously, and to elucidate when this becomes a problem. For example, the actual RFID hardware using during the course of this project allowed up to three tags to be in range at the same time. This did not pose a problem at all during map building.

Why was this not done? This is a difficult task, and probably worth a thesis in itself. A good alternative to solving this task purely mathematically is to write a software where map building can be simulated, including the possibility of varying the reader's range, etc. There was simply no time to do all this.

Difficulty? Hard, since probably good mathematical skills are required. Even if this task would be solved by using a simulator it is still classified as hard, as it is probably not trivial to write a simulator with high enough fidelity.

Smarter way of doing gradient descent. The proposed gradient descent method is fairly straight-forward and inefficient. Particularly, determining the lowest cost

direction requires a considerable amount of time. A future work could be to investigate if it is possible to remove the time consuming rotations. Furthermore, the approach of avoiding obstacles is done by a fuzzy controller, which has problems due to the reactiveness (e.g., see Section 5.3.3). Therefore, a future work could be to investigate if some other control method are better suited for this particular task.

Why was this not done? The aim and goal of this project were not to find an optimal navigation algorithm, but rather to show that navigation through a distance map is possible. The navigation algorithm discussed in this report indeed indicate this, so it was not a necessity to develop a better method.

Difficulty? Medium

Apart from the proposals above, it might be interesting to deal with the two limitations mentioned in Section 1.3.3. A possible solution (for both limitations) is to take inspiration from how ants adapt to a dynamic environment, and changing goals of a nest; pheromone evaporation [7]. In the context of what has been presented in this report, i.e., a pheromone trail represented by "costs" in a collection of RFID tags, evaporation or degradation might be achieved by associating each cost with a time stamp, describing the age of a cost (i.e., when it was stored). A future work could be to investigate to how and when such time stamps can be used to adapt to changes in the environment.

Chapter 9

Acknowledgments

I would like to thank my supervisor Prof. Alessandro Saffiotti who patiently helped me whenever needed. I am very grateful to my supervisor to give me an opportunity to perform my degree project on this very interesting research topic. I would also like to thank Per Sporrong for his great help in solving different hardware problems that occurred with the mobile robot.

Finally I would like to thank all other people at AASS who in any way helped me during the course of the project, directly or indirectly. In particular, thank you, Dr. Mathias Broxvall for your great PEIS applications.

References

[1] Homepage of AASS. Online at http://www.oru.se/aass. Accessed at 2008-03-08.

[2] Homepage of MobileRobots. Online at http://www.mobilerobots.com/. Accessed at 2008-04-07.

[3] Homepage of the PEIS Ecology Project. Online at http://www.aass.oru.se/~peis/. Accessed at 2008-04-05.

[4] Himanshu Bhatt and Bill Glover. *RFID Essentials*. O'Reilly, 2006.

[5] M. Broxvall, M. Gritti, A. Saffiotti, B.S. Seo, and Y.J. Cho. PEIS ecology: Integrating robots into smart environments. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 212–218, Orlando, FL, 2006. Online at http://www.aass.oru.se/~asaffio/.

[6] M. Broxvall, B.S. Seo, and W.Y. Kwon. The peiskernel: a middleware for ubiquitous robotics. In *Proc. of the IROS-07 Workshop on Ubiquitous Robotic Space Design and Applications*, San Diego, California, 2007.

[7] Renato R. Cazangi, Fernando J. Von Zuben, and Mauricio F. Figueiredo. Stigmergic autonomous navigation in collective robotics. In *Studies in Computational Intelligence*, volume 31, pages 25–63. Springer Publishing, 2006.

[8] K. Chong and L. Kleeman. Accurate odometry and error modelling for a mobile robot. In *International Conference on Robotics and Automation*, volume 4, pages 2783–2788, 1997.

[9] Grosan Crina and Abraham Ajith. Stigmergic optimization: Inspiration, technologies and perspectives. In *Studies in Computational Intelligence*, volume 31, pages 1–24. Springer Publishing, 2006.

[10] B. Gerkey, R. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics*, pages 317–323, Coimbra, Portugal, 2003.

[11] Texas Instruments. Datasheet for the RI-I02-112A-03 and RI-I02-112B-03 transponders. Online at http://www.ti.com/rfid/shtml/rfid.shtml. Accessed at 2008-03-08.

[12] MobileRobots. AmigoBot User's Guide. Online at http://www.mobilerobots.com/. Accessed at 2008-04-07.

[13] Kevin M. Passino and Stephen Yurkovich. *Fuzzy Control*. Addison-Wesley, 1998.

[14] A. Saffiotti and M. Broxvall. PEIS ecologies: Ambient intelligence meets autonomous robotics. In *Proc of the Int Conf on Smart Objects and Ambient Intelligence (sOc-EUSAI)*, pages 275–280, Grenoble, France, 2005. Online at http://www.aass.oru.se/~asaffio/.

[15] A. Saffiotti, M. Broxvall, B.S. Seo, and Y.J. Cho. The PEIS-ecology project: a progress report. In *Proc. of the ICRA-07 Workshop on Network Robot Systems*, pages 16–22, Rome, Italy, 2007. Online at http://www.aass.oru.se/~asaffio/.

[16] B.A. MacDonald T.H.J. Collett and B.P. Gerkey. Player 2.0: Toward a practical robot programming framework. In *Proc. of the Australasian Conf. on Robotics and Automation (ACRA)*, Sydney, Australia, December 2005.

[17] Frank Thornton, Brad Haines, Anand M. Das, Hersh Bhargava, Anita Campbell, and John Kleinschmidt. *RFID Security.* O'Reilly, 2006.

Appendix A

Logged Gradient Descent Paths

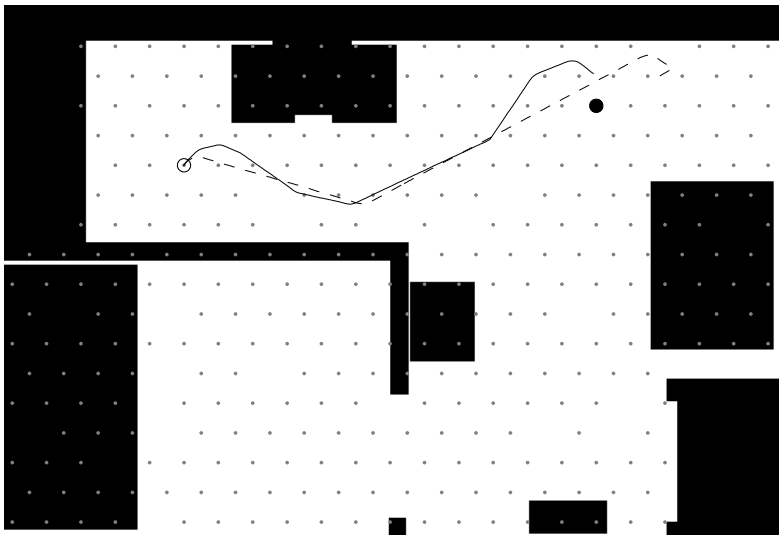
This appendix shows logged paths of tests performed in Section 7.1. Due to the amount of figures generated, and the size of each figure, placement in Section 7.1 was not appropriate.

Some general characteristics can be state for all figures shown here:

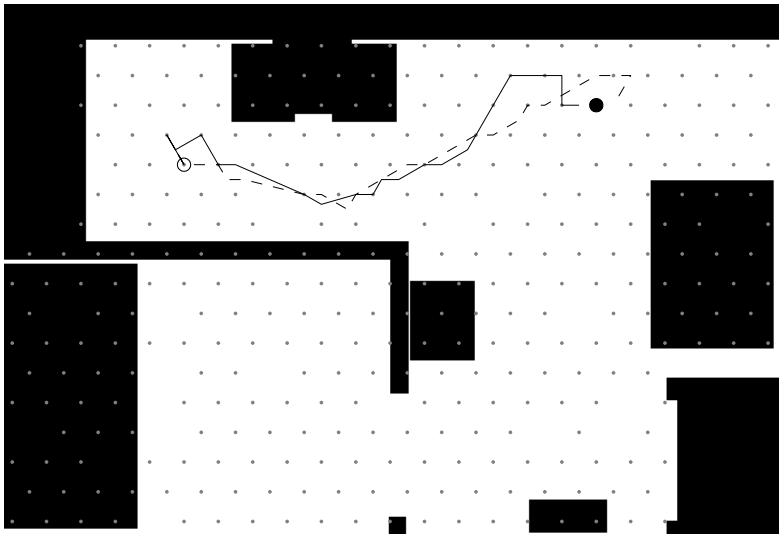
- An open black circle represents the start while a filled black circle is the goal.
- Each sub-figure contains two paths; (1) a path denoted by a solid line; (2) a path denoted by a dashed line. (1) and (2) usually are the best and the worst test runs, respectively, with respect to some measurement. Moreover, (1) and (2) are always test runs from the same test.
- Each figure contains two sub-figures. Both sub-figures show the same paths, but calculated using different position estimation methods; the above sub-figure uses odometry while the one below uses the known positions of the RFID tags.

A few comments indicating which test each figure belongs to:

- Figure A.1 and Figure A.2 belong to tests described in Section 7.1.2.
- Figure A.3 and Figure A.4 belong to tests described in Section 7.1.3.
- Figure A.5 belong to the test described in Section 7.1.4.

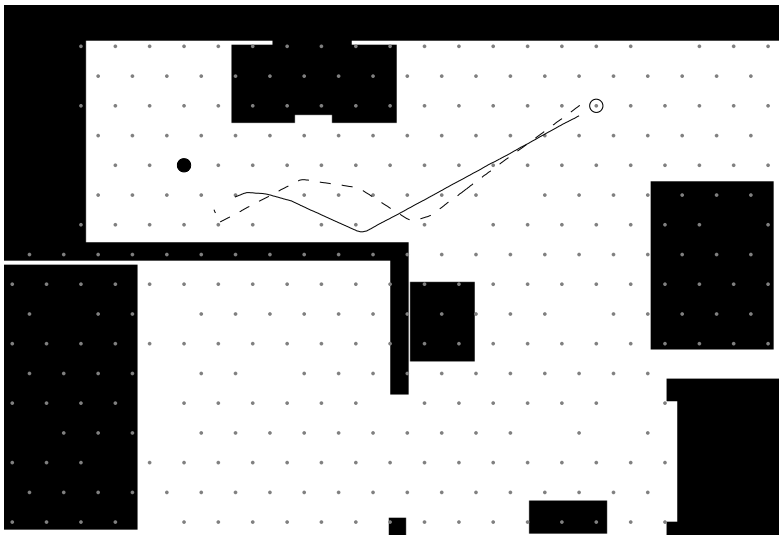


(a) Position estimation by odometry

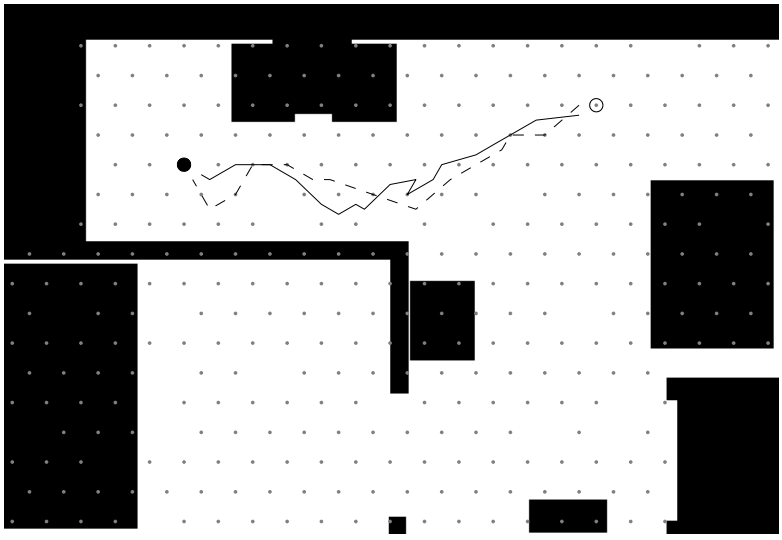


(b) Position estimation by RFID tags

Figure A.1: The above sub-figures show logged paths for the best and then the worst test run with respect to navigation time. Navigation took place from the kitchen to the living room. More information about the test can be found in Section 7.1.2.

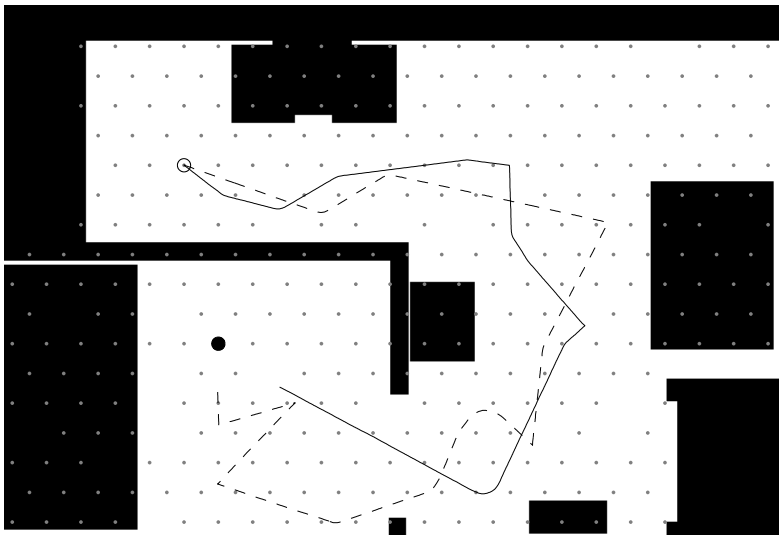


(a) Position estimation by odometry

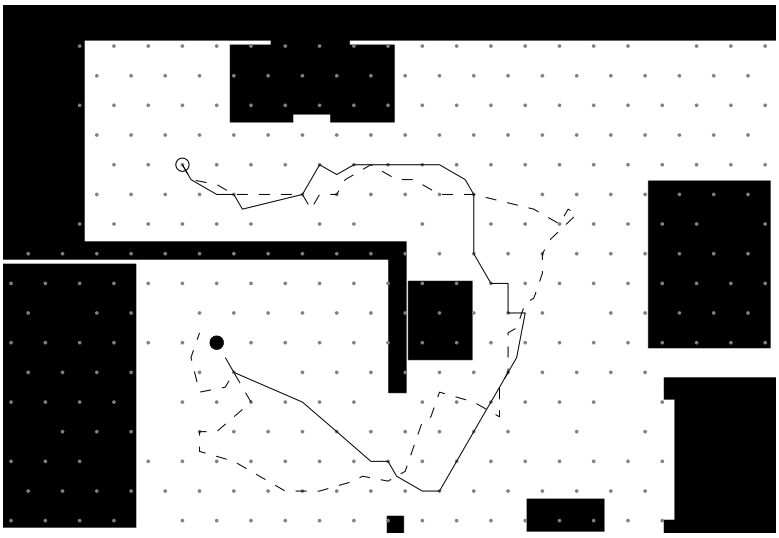


(b) Position estimation by RFID tags

Figure A.2: The above sub-figures show logged paths for the best and then the worst test run with respect to navigation time. Navigation took place from the living room to the kitchen. More information about the test can be found in Section 7.1.2.

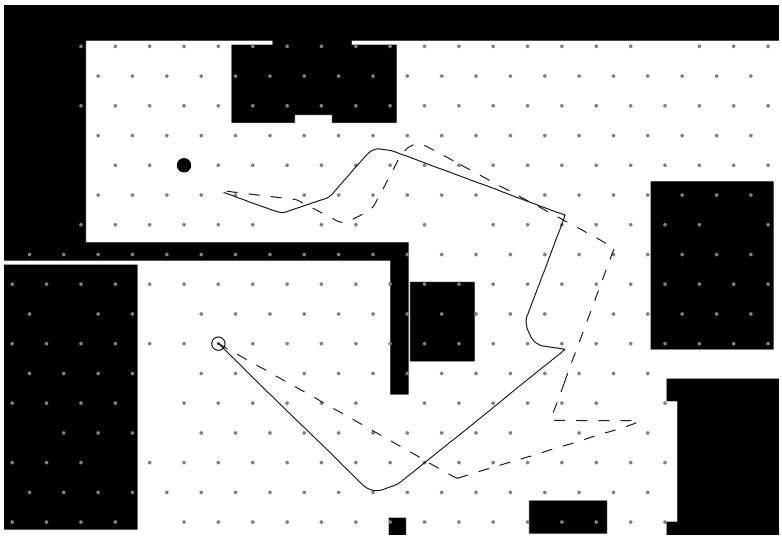


(a) Position estimation by odometry

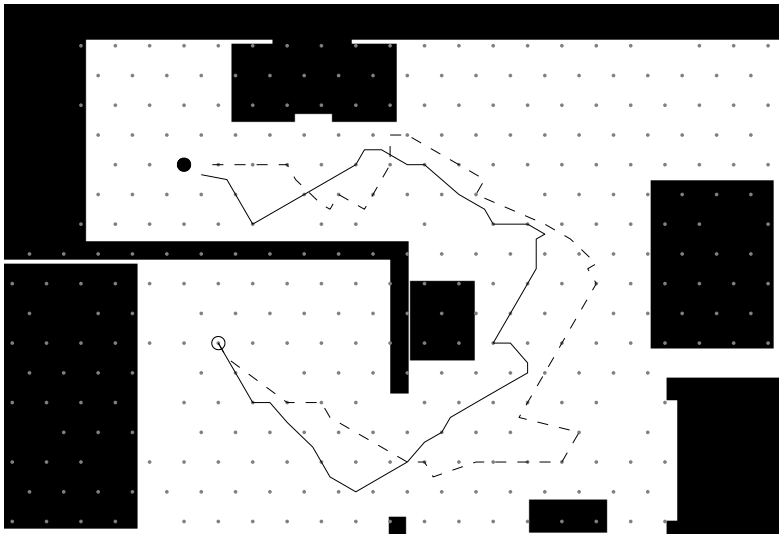


(b) Position estimation by RFID tags

Figure A.3: The above sub-figures show logged paths for the best and then the worst test run with respect to navigation time. Navigation took place from the kitchen to the bedroom. More information about the test can be found in Section 7.1.3.

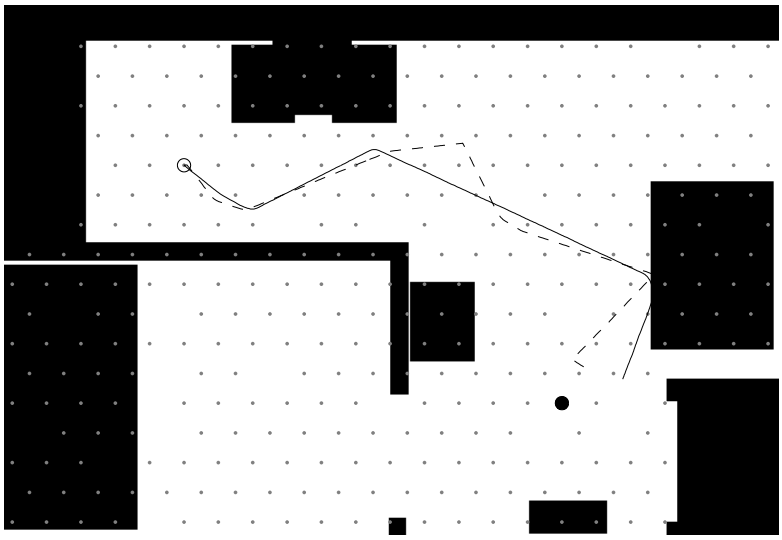


(a) Position estimation by odometry

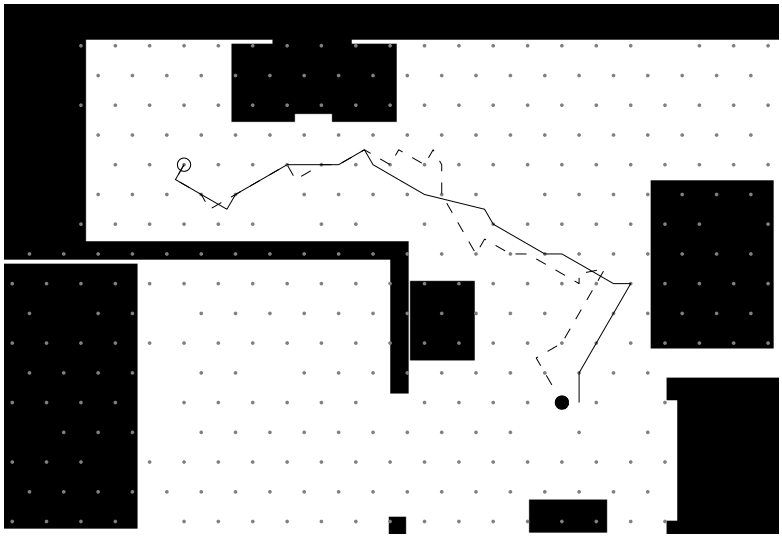


(b) Position estimation by RFID tags

Figure A.4: The above sub-figures show logged paths for the best and then the worst test run with respect to navigation time. Navigation took place from the bedroom to the kitchen. More information about the test can be found in Section 7.1.3.



(a) Position estimation by odometry



(b) Position estimation by RFID tags

Figure A.5: The above sub-figures show logged paths of two extreme test runs (with respect to the number of rotations). Navigation took place from the kitchen to the living room. More information about the test can be found in Section 7.1.4.