INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Autonomous Mobile Robot Navigation using Smartphones

André Guilherme Nogueira Coelho dos Santos

Dissertation for the achievement of the degree

Master in Information Systems and Computer Engineering

Committee

Chairman:	Professor Doutor José Carlos Martins Delgado
Supervisor:	Professor Doutor João Manuel Paiva Cardoso
Advisers:	Professor Doutor Pedro Nuno Ferreira da Rosa Cruz Diniz
	Professor Doutor Rui Fuentecilla Maia Ferreira Neves

November 2008

Acknowledgments

When working on any project, no matter how individual it can be, it will almost certainly require input, assistance or encouragement from others. My project is no exception to this rule.

First of all I would like to thank Professor João Cardoso, for his guidance, support and teaching which opened my eyes for academic research and for the embedded systems field of computer science. I would also like to thank my colleagues Luís Tarrataca and Bruno Gouveia for many interesting and mind opening conversations and input that helped throughout some tough days.

I would also like to thank Instituto Superior Técnico (IST) for infrastructure and material support as well as Professor José Delgado who was responsible for the funding (from a "Melhoria da Qualidade de Ensino" IST project) to acquire the Lego NXT robots and the Nokia N95 model smartphones used in this thesis.

I sincerely thank the donation by Nokia Finland of the N80 smartphones also used during this thesis work.

Most importantly, I would like to thank my parents, Guilherme and Maria for making it possible for me to pursuit my college ambitions and dreams, relieving me from many of life's problems as well as supporting me throughout my decisions. I have also to thank Marta and the rest of my family and friends, for making me the person I am today, for being by my side whenever I needed a friendly shoulder, a simple talk or just some time off work. I also have to say sorry, if sometimes work came first.

I certainly hope this thesis provides as good reading as the pleasure I had to write it.

Abstract

This thesis studies the feasibility of a smartphone embedded system as a navigation control unit for mobile robots. This work may contribute to the advance of the Embedded Systems field by the development of Java embedded solutions that show today's mobile phone capacity when executing complex navigation algorithms for mobile robot control.

In this work we used a system composed by a mobile robotic component (Lego's NXT Mindstorms) and a control component executed by smartphones (Nokia N80 and N95). The smartphone is the embedded system that executes navigation algorithms, which were implemented considering known mobile restrictions and limitations, e.g., memory capacity and processing speed. The algorithms implemented for Path Planning, Localization and Mapping were: Potential Fields, Particle Filters and Visual Landmark Recognition, respectively. All implementations were developed using the Java programming language mobile edition (J2ME).

The experimental results conducted, verified that, although all the hardware restrictions and the algorithms's mobile adaptation, it was still possible to execute the referred algorithms in real-time and with acceptable results. The system was able to identify landmarks, retrieve possible locations of where it could be present and successfully plan its motion avoiding obstacles.

We demonstrate that the current generation of high-end mobile phones are ready to execute such complex and demanding algorithms and therefore be a feasible solution for mobile robot control. With the current capacity, these smartphones can assume other roles and become a more essential part of our lives.

Keywords: Particle Filter, Potential Fields, Visual Landmark Recognition, Smartphone, Mobile Robotics, J2ME.

Resumo

Esta tese estuda a praticabilidade actual de um sistema embebido, smartphone, a actuar como uma unidade de controlo de navegação para robôs móveis. O trabalho executado pretende contribuir para a área de Sistemas Embebidos, através do desenvolvimento de soluções embebidas em Java, que analisem a capacidade actual da tecnologia do telefone móvel em processar e executar algoritmos complexos de navegação.

Neste trabalho foi usado um sistema composto por um componente de robótica móvel (NXT Mindstorms da Lego) e por um componente de controlo, os smartphones (Nokia N80 e N95). O smartphone é o sistema embebido que executa os algoritmos de navegação, implementados considerando as restrições e limitações dos dispositivos móveis. Os algoritmos de planeamento de percurso, localização e mapeamento implementados foram: campos potenciais, filtro de partículas e reconhecimento visual de marcas, respectivamente. Todas as implementações foram desenvolvidas na edição móvel da linguagem de programação Java (J2ME).

Através dos resultados experimentais conduzidos, foi possível verificar que, apesar da existência das limitações de hardware e da adaptação dos algoritmos, é viável a execução dos algoritmos referidos em tempo real com resultados aceitáveis. O sistema conseguiu identificar marcas, encontrar possíveis localizações para a posição actual e com sucesso movimentar-se evitando obstáculos.

Podemos comprovar que a geração actual de telefones móveis de gama alta está pronta para executar algoritmos complexos e consequentemente ser uma solução praticável para o controlo de robótica móvel. Com a capacidade actual, os smartphones podem assumir outros papéis tornando-se cada vez mais essenciais nas nossas vidas.

Palavras Chave: Filtro de Partículas, Campos Potenciais, Reconhecimento Visual de Marcas, Smartphone, Robótica Móvel, J2ME.

Contents

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface
CLDC	Connected Limited Device Configuration
CPU	Central Processing Unit
EKF	Extended Kalman Filter
ESS	Effective Sample Size
FIRAS	Force Inducing an Artificial Repulsion from the Surface of the Obstacle (in French)
GNRON	Goals Non-Reachable with Obstacles Nearby
GUI	Graphical User Interface
I/O	Input and Output
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JSR	Java Specification Request
JVM	Java Virtual Machine
LCD	Liquid Crystal Display
MIDP	Mobile Information Device Profile
NASA	National Aeronautics and Space Administration
NBC	Next Byte Codes
NQC	Not Quite C
NXC	Not eXactly C
PC	Personal Computer
PDF	Probability Density Function
RAM	Random Access Memory
RGB	Red Green Blue
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous Localization And Mapping
XML	eXtensible Markup Language
WTK	Wireless ToolKit

Chapter 1

Introduction

"Hell, there are no rules here - we're trying to accomplish something."
Thomas Edison

The world of Robotics is one of the most exciting areas that has been through constant innovation and evolution. Robotics are interdisciplinary and have become more and more a part of our lives. They are no longer a vision for the future but a reality of the present. In our day to day lives, robots fill many important parts from mowing the lawn, securing our uninhabited houses, feeding our pets or building our cars. It is visible that they are everywhere and as time passes they will be more and more present.

Within robotics, special attention is given to mobile robots, since they have the capability to navigate in their environment and are not fixed to one physical location. The tasks they can perform are endless and the possibilities for them to contribute to our lives are countless (e.g., unmanned aerial vehicles, autonomous underwater vehicles).

In recent years we have witnessed a major increase of mobile robots. Nowadays they are subject of major research projects being present in industry, military, security and even home environments as consumer products for entertainment and home aid tasks. In Figure 1.1 mobile robot examples are presented in chronological order that illustrate important milestones in the mobile robotics history.

As technology evolves, many developments, also due to mobile robotics, have been achieved on embedded systems. Hardware is shrinking in size and increasing in capabilities providing more and more computational power. Mobile phones are examples of that evolution. From heavy, expensive and with poor capabilities to small, cheap and rich smartphone platforms, mobile phones are today one of the most used embedded systems in the world. Today, a mobile phone is a fully capable small computer with great communication capabilities, powerful processing, built-in camera, sound player, movie player and much more. In Figure 1.2, examples of the mobile phones evolution are presented.

From both the mobile robotics evolution and the mobile phone evolution we can see that Moore's Law[1] is in evidence in the embedded systems market. Year by year we witness significant improvements in both

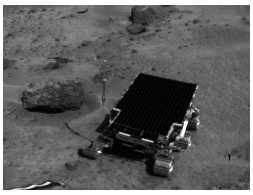
[1]Moore's Law states that the number of transistors on a chip will double about every two years.

(a) Bristol Tortoise by W. Grey Walter (1948-1949)

(b) Mowbot by Mowbot Inc. (1969)

(c) Hero by Heathkit (1980)

(d) Dante II by Carnegie Mellon University (1993-1994)



(e) Sojourner Mars Rover by NASA (1996-1997)

(f) Asimo by Honda Motor Company (First introduced in 2000)

(g) Roomba by iRobot (2002)

(h) Stanley by Stanford University's Stanford Racing Team in cooperation with the Volkswagen Electronics Research Laboratory (2005)

Figure 1.1: Mobile Robot Examples.

(a) Military Portable Phone

(b) Portable Phone

(c) Motorola DynaTAC 8000X (1983)

(d) Nokia E90 Communicator (2007)

Figure 1.2: Mobile Phone Examples.

areas which grow at enormous speed. It is in this constant embedded evolution environment, that this thesis focuses on a joint system with mobile robotics and smartphones, aiming at contributing to the Embedded Systems field by analyzing the performance of smartphones when executing mobile robotics navigation algorithms.

1.1 Challenges

The mobile robotics field has many challenges that need to be addressed. Navigation (mapping, localization, path planning), environment perception (sensors), autonomous behavior (actuators, behavior rules, control mechanisms), two legged movement (balance), capacity to work for an extended period without human intervention (batteries, human assistance), are research opportunity examples, among many others. Some of these challenges are addressed in this thesis.

1.2 Objectives

This thesis focuses on the development of a navigation system composed by a mobile robot and a smartphone. We are motivated in providing mobile phone solutions to the main areas in mobile robot navigation. In this system, the mobile robot is controlled by the smartphone, where navigation algorithms are processed, generating controls that are transmitted back to the mobile robot using Bluetooth communication.

Figure 1.3 shows the structure of the final system. The robot platform used is Lego's NXT Mindstorms [Lego, 2008b], a versatile well equipped set that enables fast and easy construction of different prototypes. For the smartphone, Nokia's N80 [Nokia, 2008a] and N95 [Nokia, 2008b] mobile phones are used. Both Nokia models are very powerful smartphones for today's standards providing us with good performance, capacity and functionality.

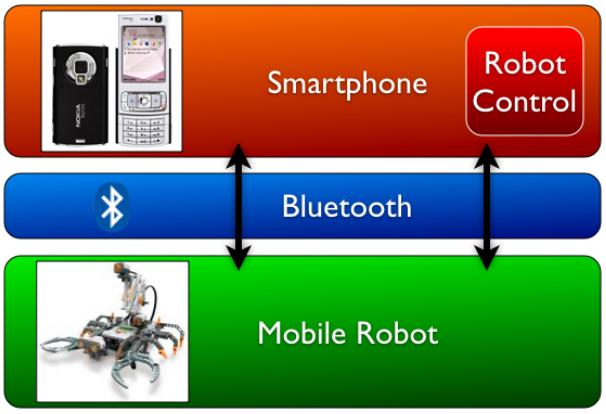


Figure 1.3: System architecture schematic.

Within the scope of this system, the work presented has the following main objectives:

- Create a prototype with the architecture provided in Figure 1.3, which includes software development for both the smartphone component and the mobile robot component, as well as for the communication layer;
- Research and develop navigation algorithms for mobile robot control, considering their implementation in J2ME to smartphone embedded systems;
- Study the potential of code optimizations for overall performance improvement on smartphones;

- Study and analyze smartphone capacity and performance when executing computationally demanding navigation algorithms;
- Research and evaluate the overall feasibility of a smartphone, as an image capture and processing unit for visual landmark recognition.

1.3 Contribution

The main contribution of this thesis is the study of the current feasibility of smartphones as devices responsible for mobile robot navigation. This study includes:

- Study of interaction and integration solutions for joint systems between different embedded devices, in this case, between mobile robots and smartphones.
- The programming and development of navigation algorithms, which are computationally demanding and complex, for smartphones, which are computationally limited devices.

1.4 Thesis Organization

This thesis addresses a number of topics of mobile robot navigation. For better understanding of these topics, we start by introducing the main concepts about mobile robotics navigation and by presenting the algorithms used in this work. The implemented algorithms are individually tested and experimental results are given for each. In the final chapters the work is discussed by presenting the prototype's experimental results and the conclusions drawn.

- **Chapter 2 - Mobile Robot Navigation**
  Provides an introduction to mobile robot navigation, by presenting three navigation problems and robot control architectures. Each navigation problem is introduced with examples of known approaches. Motivational examples and applications are also enumerated to demonstrate the importance of mobile robot navigation.
- **Chapter 3 - Mapping using Visual Landmark Recognition**
  Mapping can be accomplished through many procedures. Using vision sensors is one possible solution. This chapter concentrates on a vision based landmark recognition solution. A simple approach considering artificial landmarks is introduced with each step explained, and with distance and orientation test results.
- **Chapter 4 - Localization using Particle Filter**
  Chapter that presents more topics on Localization and explains the Particle Filter approach. The developed mobile implementation of the Particle Filter method is also presented, with experimental results and conclusions obtained from simulations conducted.
- **Chapter 5 - Path Planning using Potential Fields**
  This chapter concentrates on the navigation path planning method of potential fields. The mathematical model is introduced as well as the inherited problems of this method. We also present solutions for the

referred problems and discuss the implementation of a mobile version for the Potential Fields approach. Simulations of this method are also presented.

- **Chapter 6 - Embedded System Prototype**
  All implementation related aspects are presented in this chapter. Both the mobile robot and smartphone components are explained and justified, as well as the tools and technology used. The software development cycle applied is also shown. For better and easier interaction between the two components, we present a developed a middleware solution.

- **Chapter 7 - Experimental Results**
  The Experimental Results chapter presents profiling and testing on the field. Results from experiments with the three navigation problems addressed are illustrated.

- **Chapter 8 - Conclusions**
  This chapter resumes the most important aspects of the work completed, the overall relevance of the subjects approached and presents future work that could improve the current system.

- **Appendix A - Potential Fields XML Input**
  This appendix introduces the XML file format that is used to input data onto the potential fields algorithm.

- **Appendix B - Prototype Hardware Specification**
  This appendix presents the hardware specification for the prototype developed. The specification includes both smartphones and the mobile robot kit.

Chapter 2

Mobile Robot Navigation

"One, a robot may not injure a human being, or through inaction, allow a human being to come to harm; Two, a robot must obey the orders given to it by human beings except where such orders would conflict with the First Law; Three, a robot must protect its own existence as long as such protection does not conflict with the First or Second Laws."

Laws of Robotics by Isaac Asimov

2.1 Introduction

A robot is an autonomous system that can be able to sense its environment, and to act on it to achieve goals. A mobile robot adds the fact that it is not confined to one specific location, as it has the capability of moving in its environment. The main characteristic that defines an autonomous robot is the ability to act on the basis of its own decisions, and not through the control of a human [Matarić, 2007].

Navigation is defined as the process or activity of accurately ascertaining one's position, planning and following a route. In robotics, navigation refers to the way a robot finds its way in the environment [Matarić, 2007] and is a common necessity and requirement for almost all mobile robots.

Robot navigation is a vast field and can be divided into subcategories for better understanding of the problems it addresses. Leonard and Durrant-Whyte [Leonard and Durrant-Whyte, 1991], briefly described the general problem of mobile robot navigation by three questions, each one referring a subcategory:

- **"Where am I?"**
  This question constitutes the **localization** problem which can be stated as: A robot is at an unknown position in an environment for which it has a map. It "looks" about its position, and based on these observations it must infer the place (or set of places) in the map where it could be located [Guibas and Motwani, 1995].
- **"Where am I going?"**
  The **mapping** problem consists in a robot being in an unknown environment which it does not have a map for, and by navigating through it enables the creation of a map representation. The robot must identify special features in environment landmarks in order to acknowledge where obstacles are located.

- **"How do I get there?"**
  The **path planning** problem states a general concern with finding paths connecting different locations in an environment according to specific objectives and constraints.

Navigation is today a known difficult and complex problem, with many different solutions and approaches. So, why does robust and reliable autonomous mobile robot navigation remains such a difficult problem? In [Leonard and Durrant-Whyte, 1991] is referred that it is not the navigation process per se that is the problem, it is the reliable acquisition or extraction of information, and the automatic correlation or correspondence of that information with some navigation map, that makes the autonomous navigation problem so difficult.

2.2 Motivational Examples and Applications

Mobile robot navigation is focus of growing attention. Taking into account the progress in technology, we can perceive that the future for mobile robotics will be very bright and navigation will certainly have a primary role in that future.

To better understand the wideness of applications where this subject can be applied and bring benefits to, some of today's usage of mobile robotics and navigation are listed bellow:

2.2.1 Driving Assistance Systems

One important application for navigation is autonomous/semiautonomous driving to assist humans. Assisted driving can help human drivers in day to day driving, aiding in many situations (e.g., when the driver is not on his full capacities, on the correction of errors in trajectories or on parallel parking). This assisted driving capability can in a more optimistic view withdraw completely the driving responsibility from the human driver, help decrease the number of road accidents and reduce traffic congestion.

There exist numerous projects that research on autonomous/semiautonomous driving:

- **Tartan Racing** [Carnegie Mellon, 2008]
  Tartan Racing Technology is responsible for project Boss, a Chevy Tahoe that autonomously navigates through normal city traffic. The car uses perception, planning and behavioral software to reason about traffic and take appropriate actions while proceeding safely to a destination.
- **Potential Fields for Lane Keeping and Vehicle Control** [Stanford, 2008]
  This project is from the responsibility of the Mechanical Engineering Department from Stanford University. The objective of this research is to incorporate collision avoidance into driver assistance systems.

2.2.2 Space Exploration Vehicles

Space Exploration Vehicles are a result from the advances achieved in mobile robot technology. Autonomous mobile robots with highly stable navigation systems provide the capabilities necessary for space exploration vehicles to have autonomy to explore the terrain, make choices and move in the unknown environment while transmitting information back to Earth.

- **Mars Exploration Rover Mission** [NASA, 2008]
  NASA's Mars Exploration Rover Mission uses technology for autonomous planetary mobility in order to enable the rovers to make decisions and avoid hazards on their own. Navigation has helped the rovers avoid mission barriers and become more capable of accomplishing their missions.

2.2.3 Military Vehicles

The United States Department of Defense has made several investments for many military autonomous robot projects. These investments have the objective of developing technology that can be used in the near future in many situations where military vehicles are needed.

- **TALON Military Robots** [Foster-Miller Inc., 2008]
  TALON military robots are powerful, durable, lightweight tracked vehicles that are widely used for explosive ordnance disposal, reconnaissance, communications, hazmat, security, defense and rescue. They have all-weather, day/night and amphibious capabilities and can navigate virtually in any terrain.

From aid providers to natural disasters or participants in peace reinforcement actions, autonomous military mobile robots can accomplish tasks which do not need to be done by humans. Human casualties can be reduced and military personal can focus their efforts on other tasks where they are more needed.

2.2.4 Home Aid Robots

Another important utility for mobile robot navigation is as home aid robots. These home aid robots can be used to accomplish difficult and monotonous tasks that do not need to be done by humans, leaving humans with more time to accomplish other tasks. Also, these robots can be specialized and optimized to accomplish these tasks with the very best results. From cleaning floors, gutters and pool tiles to mowing the lawn, the applications are endless.

- **iRobot** [iRobot Corporation, 2008]
  The iRobot company delivers innovative robots that can make a difference in peoples day to day lives by helping with domestic dirty work (e.g., cleaning floors, gutters, pools and mowing the law).

2.3 Robot Control Architectures

Autonomous robots may be operated by different schemes. A robot control architecture provides rules, guiding principles and constraints for organizing a robot's control system, programs and control algorithms so it can be autonomous and achieve goals.

There are different types of control architectures [Matarić, 2007] which differ in speed, modularity and representation:

- Deliberative Control;
- Reactive Control;
- Hybrid Control;

- Behavior-Based Control.

2.3.1 Deliberative Control

Deliberative Control consists in a long-term control which concentrates efforts thinking in decision and action planning. It uses a simple sequence of three steps: sense, plan and act (Figure 2.1). By first sensing the environment, necessary data for calculating possible action outcomes is gathered. This process of searching throughout possible outcomes and choosing what best achieves a goal is called planning. Once a decision is made, an action can be done.

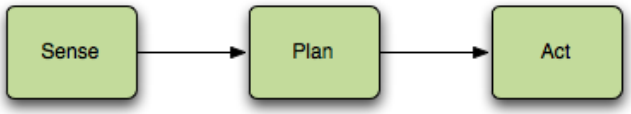


Figure 2.1: Deliberative "Sense-Plan-Act" architecture.

Although better understanding the environment and being able to plan a potentially better action, due to the possible complexity of this type of control, real-time control is sometimes complicated, making it difficult for a robot to navigate in dynamic environments. Computation time, large amounts of data and complex processing are therefore the main drawbacks of Deliberative Control.

2.3.2 Reactive Control

Reactive Control consists on systems that navigate in real-time reacting to current sensorial information that is perceived from the environment. In this approach, complex computation is exchanged by fast, stored, pre-computed responses. Stimulus are sensed from the environment and response actions are produced based on the programmed robot behavior (see Figure 2.2).

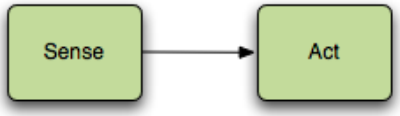


Figure 2.2: Reactive "Sense-Act" architecture.

2.3.3 Hybrid Control

Hybrid Control involves the combination of the reactive control and the deliberative control architectures. The idea is to gather the best of both approaches: the speed of the reactive control and the intelligence of the deliberative control. The control is based on three layers, the planning layer, the reactive layer and a middle layer that exists to guarantee the safe and consistent interaction of the other two layers, as shown in Figure 2.3.

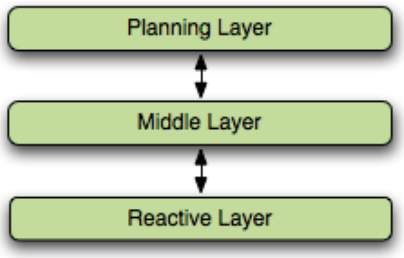


Figure 2.3: Hybrid "three layer" architecture.

The primary problem of this approach is the coordination of the planning and reactive layers. Those layers have very different characteristics and need to be consistent and robust in order to produce real-time valuable results.

2.3.4 Behavior-Based Control

Behavior-Based Control incorporates the best of reactive systems and some deliberation, but does not involve a hybrid solution. In this type of control, actions are made according to behaviors, that can be anything a robot can do. Behavior-Based Control can achieve real-time navigation allowing for state and environment representation, as well as the possibility of learning.

2.4 Information Representation

Mobile robots need to remember information about the environment they navigate in. The process of navigation involves managing great quantities of data that need to be stored, accessed and used to provide accurate motion.

Representation refers to the way information is stored or encoded in the robot and can take a great variety of forms. A robot can represent information about itself, other robots, objects, obstacles, people, the environment, tasks and actions. As time passes, representation may need to be updated, thus having computational and memory costs to the robot.

Recognizing the important role information representation takes in mobile robot navigation, special attention has to be made when storing data, choosing what to store, for how long and why to store it. For instance, information from the environment surroundings gathered will influence the quality and speed of navigation.

For the mobile robot it is a requisite that representation models be easily accessed in real-time while also keeping memory usage low.

2.5 Localization

Localization is the process of estimating where the robot is, relatively to some model of the environment, using whatever sensor measurements are available. As the robot keeps moving, the estimation of its position drifts and changes, and has to be kept updated through active computation [Matarić, 2007]. These updates are performed based on the recognition of special features in landmarks, sensor data and probabilistic models.

Robot localization is a key problem in making truly autonomous robots. If a robot does not know where it is, it can be difficult to determine what to do next. In order to localize itself, a robot has to access relative and absolute measurements which return feedback about its driving actions and the situation of the environment. Given this information, the robot has to determine its location as accurately as possible [Negenborn, 2003]. Uncertainty rises from the sensing of the robot because of three main aspects:

- The estimation process is indirect;
- The measurements are noisy and problematic because of real-world sensors;
- The measurements may not be available at all times.

Based on the uncertainty characteristics of the problem, localization as other important mobile robotics problems, has been tackled by probabilistic methods [Thrun et al., 2005], being the most commonly used:

- Markov Localization [Fox, 1998]
- Particle Filters [Gordon et al., 1993]

2.5.1 Markov Localization

Markov Localization [Fox, 1998] is a probabilistic method that estimates a robot's location from sensor data, providing accurate position estimates in dynamic environments. The key idea of Markov Localization is to maintain a probability density over the space of all possible locations of a robot in its environment.

In order to better understand the probabilistic nature of localization, consider the Equations 2.1 (belief prediction) and 2.2 (belief distribution) and the situations in Figures 2.4, 2.5, 2.6 and 2.7. The example scenario, for simplicity purposes, considers only a one-dimension space where the robot can move along an axis (robot state x). For other dimensions the process is analogous. Each situation is composed by an image and the graphical representation of the belief distribution ($Bel(x)$).

The belief reflects the robot's internal knowledge about its state (x_t) and its distribution probabilities to each possible state. Updates to the belief are made when measurements z_t in the environment are performed, providing necessary data for the motion model for robot control (u_t).

$$\overline{Bel}(x_t) = \int p(x_t|u_t, x_{t-1}, m)\ Bel(x_{t-1})\ \partial x_{t-1} \tag{2.1}$$

$$Bel(x_t) = \eta\ p(z_t|x_t, m)\ \overline{Bel}(x_t) \tag{2.2}$$

The top of Figure 2.4 shows a robot somewhere in an environment without knowing its location. Markov

localization represents the level of uncertainty of the robot's localization by an uniform distribution over all possible positions (belief is uniform), as shown by the graph (bottom of Figure 2.4). At this stage, all possible locations have the same probability (i.e., the robot assumes it can be everywhere).

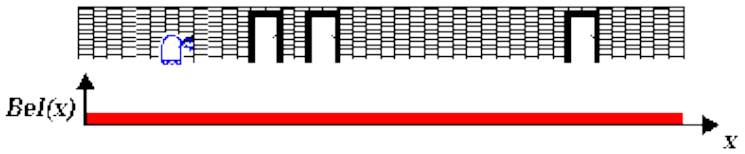


Figure 2.4: Markov Localization situation 1 (based on [Fox et al., 1999]).

Now let us assume the robot uses its sensors and captures the information representing that it is next to a door (see Figure 2.5). Markov localization updates the belief distribution by raising the probability for places that are next to doors, and lowering it anywhere else (initial belief is updated). This is illustrated on the bottom of Figure 2.5. Notice that now, not all positions have the same probability. At this point in time, the robot is most likely in some locations than others. The probabilities away from positions that are not doors are not zero, this is because of noisy sensor readings that need to be taken into account, and a single sight of a door is typically insufficient to exclude the possibility of not being next to a door.

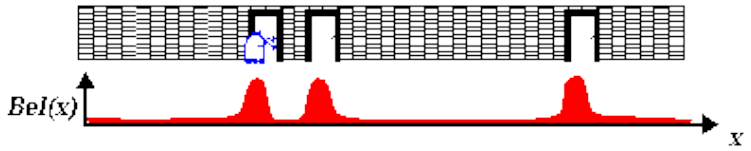


Figure 2.5: Markov Localization situation 2 (based on [Fox et al., 1999]).

Now let us assume the robot moves a meter forward. Markov localization incorporates this information by shifting the belief distribution accordingly (convolving its belief with the motion model), as depicted in Figure 2.6. To account for the inherent noise in robot motion, which inevitably leads to a loss of information, the new belief is smoother and less certain than the previous one.

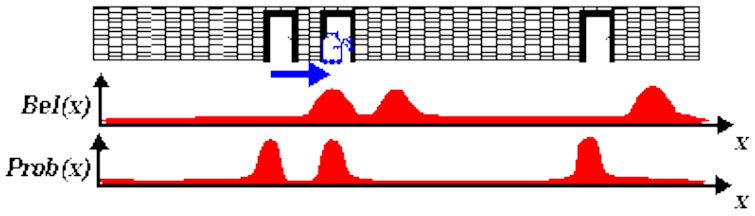


Figure 2.6: Markov Localization situation 3 (based on [Fox et al., 1999]).

Finally, let us assume the robot senses a second time, and again it finds itself next to a door. This second measurement allows to correct the previous prediction leading to the final belief shown in Figure 2.7. At this point in time, most of the probability is centered around a single location. The robot is now quite confident about its position next to the second door.

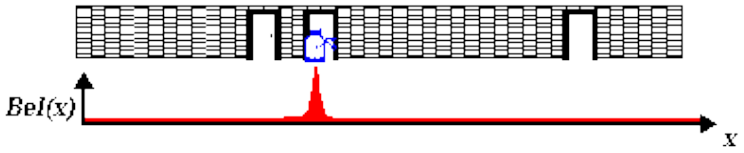


Figure 2.7: Markov Localization situation 4 (based on [Fox et al., 1999]).

2.5.2 Particle Filters

Particle Filters [Gordon et al., 1993] are sophisticated model estimation techniques based on simulation. The Particle Filter method approximates the sequence of probability distributions of interest over time using a large set of random samples, named particles. These particles have associated a weight which describes the quality of the specific particle. In contrast to the widely used Kalman Filters [Kalman, 1960] which use gaussian distributions, particle filters can approximate a large range of probability distributions.

This approach is very popular because of being relatively easy to implement and adapt to the available computational resources and localization necessity at hand, since the number of particles can be dynamically adjusted to each situation. Particle Filters have been greatly used to solve localization problems. One example is present in [Dellaert et al., 1999].

2.6 Mapping

The mapping problem exists when the robot does not have a map of its environment and incrementally builds one as it navigates. This navigation problem can be decomposed in two variants, one where the robot knows the location of the target goal and another where the target goal is not known.

In the first case, the robot can move towards the goal and meanwhile identify landmark features. This identification enables the robot to create a representation of the environment as it goes along to the target goal, planning its path and avoiding obstacles. In the second case, the robot must navigate through the environment searching for the target goal location and recognizing obstacles in order to avoid them. Both the target goal and the obstacles need to be identified by special distinctive landmark features. In the worst case, to find the target goal the robot must search the entire environment.

Of course, not all mapping problems are equally hard. The hardness of the mapping problem is the result of a different number of factors, the most important of which are [Thrun et al., 2005]:

- **Size of the environment**: The larger the environment relative to the robot's perceptual range, the more difficult it is to create a map from it.
- **Noise in perception and actuation**: If robot sensors and actuators were noisefree, mapping would be simple. The larger the noise in both perception and actuation, the more difficult the mapping problem becomes.
- **Perceptual ambiguity**: If different locations are much alike, it becomes complicated to distinguish those different locations.

- **Cycles**: Cycles in the environment are particularly difficult to map. Several runs in a corridor can correct odometry errors incrementally, but cycles make robots encounter different paths, and when closing the cycle the accumulated odometric error can be high.

The main concern for the mapping problem is how does the mobile robot perceive the environment. To locate obstacles, detect distances, observe landmarks, etc, the mobile robot must have a sensing mechanism that enables taking measurements. The sensors used for mapping depend on the type of mapping that needs to be done. Most common sensors are sonar, digital cameras and range lasers.

2.7 SLAM (Simultaneous Localization and Mapping)

SLAM (Simultaneous Localization and Mapping) [Smith et al., 1990] involves both navigation localization and mapping and constitutes a technique used by robots and autonomous vehicles to build up a map within an unknown environment while at the same time keeping track of their current position. In other words, the robot does not have a map or knows where it is, and simultaneously builds a map of the environment and computes its location.

This approach is complex since it involves both localization and mapping processes, both with uncertainties associated. One main concern in SLAM is keeping the uncertainty, for both robot position and landmark position, controlled, in order to lower errors as much as possible. For this double uncertainty, SLAM usually uses Kalman Filters and Particle Filters methods.

SLAM approaches are very important and have been widely used in mobile robotics, as they address at the same time two of the most important problems in navigation.

2.7.1 Extended Kalman Filter (EKF)

The Kalman Filter [Kalman, 1960] is a recursive solution to the discrete-data linear filtering problem. It consists of a set of mathematical equations that provides an efficient computational (recursive) method to estimate the state of a process, in a way that minimizes the error. The filter supports estimations of past, present and even future states even when the precise nature of the modeled system is unknown.

The Extended Kalman Filter (EKF) is a modified approach to the regular Kalman Filter where the process to be estimated and the measurement relationship to that process is non-linear [Welch and Bishop, 2006]. In terms of computational complexity, EKF is a computationally demanding algorithm with $O(K^2)$, where K is the number of environment features.

The EKF has been successfully applied in many applications, like missions to Mars, and automated missile guidance systems [Negenborn, 2003].

2.7.2 FastSLAM

FastSLAM is an efficient SLAM algorithm that decomposes the SLAM problem into a robot localization problem, and a collection of landmark estimation problems that are conditioned on the robot pose estimation

[Montemerlo et al., 2002]. FastSLAM implementations use a combination of EKF and Particle filters, one for landmark estimates and the other for the robot position estimation.

The algorithm complexity is $O(N \log M)$, being N the number of particles and M the number of environment features, which is better than a pure EKF SLAM ($O(K^2)$). The complexity of the algorithm makes feasible the support of a very high number of landmarks in the environment.

2.8 Path Planning

Path Planning is the process of looking ahead at the outcomes of the possible actions, and searching for the sequence of actions that will drive the robot to the desired goal [Matarić, 2007]. It involves finding a path from the robot's current location to the destination.

The cost of planning is proportional to the size and complexity of the environment. The bigger the distance and the more obstacles, the higher the cost to the overall planning. The cost of planning is a very important issue for real-time navigation needs, and the longer it takes to plan, the longer it takes to find a solution. Path Planning techniques for navigation can be divided into two subcategories:

- **Local Path Planning** are solutions that do not imply much complexity since they use only local information of the environment. Although not complex, they often do not offer optimal solutions and also have the common problem of local minima.
- **Global Path Planning** takes into account all the information of the environment at the same time. Unfortunately this type of planning is not appropriate for real-time fast obstacle avoidance because of the processing needed for all the environment's data.

There are many different approaches to path planning which try to solve the problem using different techniques. Two relevant Path Planning techniques are:

- Artificial Potential Field [Khatib, 1986]
- Ant Colony [Dorigo, 1992]

2.8.1 Artificial Potential Field

The Artificial Potential Field approach [Khatib, 1986] is a local path planner method that was introduced by Khatib [Khatib, 1986]. This method defines obstacles as repelling force sources, and goals as attracting force sources. The path is then influenced by the composition of the two forces, which produces a robot motion that moves away from obstacles while moving towards the target goal.

The approach is mathematically simple and is able to produce real-time acceptable results for collision avoidance even in dynamic environments. The most known limitation of this approach is the local minima, which refers to locations that trap the robot and prevent it from reaching the target goal location. This main problem has been addressed by many different techniques that try to solve or at least minimize its impact. These techniques are described in Chapter 5.

2.8.2 Ant Colony

The Ant Colony [Dorigo, 1992] approach is a global path planner based on ant population and ant colony behaviors. This approach simulates the characteristic real behavior of ants that by nature are capable of cooperating in finding shortest path solutions from a food source to the nest without using visual cues. Also, they adapt extremely well to changes in the environment [Liu et al., 2005].

The ant behavior is possible because ants release a secretion which is called pheromone[1] on its path road, and the pheromone influences other ants path selection. When a path is selected by many ants, the pheromone on the path becomes more and more present, therefore more ants select it. As time passes the pheromone will also disappear progressively.

2.9 Summary

We give in this chapter a brief introduction to mobile robot navigation. Concepts related to the subject are introduced and the subcategories are enumerated. For better understanding of how important mobile robot navigation is, motivational examples and applications are shown. Navigation techniques for Path Planning, Mapping, Localization and SLAM are briefly described and given examples of state-of-the-art methods.

[1] A chemical substance produced and released into the environment by an animal, esp. a mammal or an insect, affecting the behavior or physiology of others of its species.

Chapter 3

Mapping using Visual Landmark Recognition

"A picture is worth a thousand words."
Proverb

3.1 Introduction

Mapping considers the problem when a mobile robot is not provided with an a priori map of its environment. Also, sometimes even when it is supplied with an environment's map, usually an architectural blueprint, these blueprints throughout time lose accuracy, as they do not accommodate changes in furniture and other items that, from a robot's perspective, determine the shape of the environment just as much as walls and doors. The capacity of being able to learn a map from scratch can reduce the efforts involved installing and preparing a mobile robot for a new environment. This capacity also enables robots to adapt to changes in the environment without human supervision [Thrun et al., 2005].

For all mapping algorithms, there has to be some perceptual sensor that is able to retrieve information from the environment. In this chapter we concentrate on a computer vision sensor, which has the responsbility of recognizing landmarks in the environment.

For example, while in movement throughout an environment, a mobile robot equipped with visual sensors will perceive numerous characteristics from the environment which will enable it to recognize its surroundings. The visual sensors will capture raw images, which are an information rich set of data. Although these images are not noise free, they contain numerous significant points of interest that can be extracted and used to associate static images to real existing locations and therefore recognize the surroundings of the current location.

As a visual sensor, our attention goes to digital cameras which are today very capable, and relatively cheap, common devices. They will be the sensors responsible for capturing an image representation of the environment.

The need for visual landmarks as aid for mapping and localization comes from the fact that errors from odometry[1] need to be corrected for more precise localization. By recognizing the location in the environment due to the observation of landmark features, problems such as wheel slippage and terrain irregularities that affect odometry measurements can be minimized.

It is important to understand that the visual perception system is inherently uncertain, due to sensor limitations, noise and complex environments. The perception of a location is therefore probabilistic, because it involves a localization guess from a static observation, which has a level of uncertainty associated with it.

3.2 Landmark Features

In order to identify locations within an environment, a mobile robot needs to observe characteristics in its surroundings that allow the inference of the location where it is currently on.

A **feature** denotes a piece of information which is relevant in an image for solving the computational task of mapping or localization. Features can be extracted from **landmarks** that are geographic characteristics easily identifiable and associated to a specific location or object.

The appearance of a landmark varies from one observation to the next. Variations due to illumination change or external clutter make it difficult to recognize the landmark. The recognition therefore assumes a probabilistic status which exists due to uncertainty in detecting landmark features. For better recognition, landmark features must be highly distinctive, allowing correct identification with low probability of mismatch. A good feature should be tolerant to image noise, changes in illumination, scaling, rotation and viewing direction.

Features can be gathered from either artificial or natural landmarks, both with advantages and disadvantages. The ability to detect natural landmarks by computer vision methods suffers limitations because of problems related to changing illumination, occlusions and shadows, but can be used in any location setting. On the other hand, the use of artificial landmarks as visual cues can improve operational characteristics of vision-based methods since they can present easy identifiable characteristics. The disadvantage is that the environment has to be engineered, which in turn limits the flexibility and adaptability to different operational sites [Baczyk et al., 2003].

The choice of a particular type of landmark has to be considered taking into account the task that the mobile robot will need to accomplish.

3.2.1 Artificial Landmarks

Artificial landmarks are specially designed markers that are placed mainly in indoor environments that permit easy identification. These markers can be difficult to plan since their aspect and position in the environment will be directly responsible for the quality of their identification. They are usually used in small space environments and can be guaranteed to be globally unique. Although advantageous for identification, the

[1] Odometry is the study of position estimation during wheeled vehicle navigation.

necessity to prepare the environment in advance limits some of its usage and flexibility on other locations rather than indoor environments.

Most common uses of artificial landmarks are of two types:

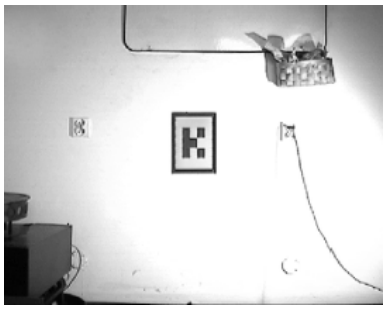
- **Solid-Color Landmarks**
  Artificial single colored shapes, easily identifiable that can be simply prepared. For better results, the colors used in the landmarks must not be easily matched by colors existing in the environment, thus helping the detection of the specific color without being confused with the background. By observing and analyzing the distribution of colors in an image, the presence of a landmark color can be assessed therefore identifying a specific landmark feature.

  In [Prasser and Wyeth, 2003] a single colored cylinder is used as a landmark with good results. Their vision system locates colored landmarks in an indoor environment measuring also the uncertainty of the observation, as well as the distance and angle to the detected landmark.

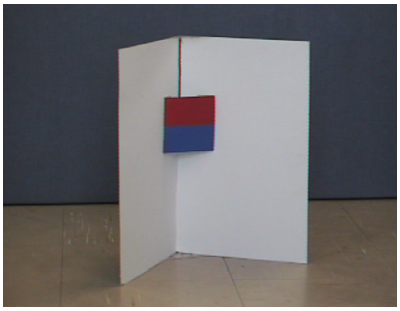
- **Patterned-Based Landmarks**
  Artificial markers with color patterns are less confused with the background environment and can also be easily identified. Pattern-Based features can be used to distinguish a large number of landmarks that can be associated to more locations.

  In the work presented in [Baczyk et al., 2003] artificial patterned features consisting on a pattern code were used for localization. The pattern consisted on a strong black orthogonal frame, filled with fluorescent green color with a chessboard (3×3) placed in the middle (see Figure 3.1(a)). Particular fields of chessboard are black or green according to the unique code, proper to the given landmark. This way, one can place a large number of unique landmarks in the workspace of the robot.

  In [Jang et al., 2002] another approach with color patterned landmarks is also used. A pattern composed by two vertically neighbor colors positioned inside a frame of two supporting panels (see Figure 3.1(b)) is used to uniquely identify different landmarks. Different colors can be used to identify more landmarks, as long as the colors used are easily differentiated.



(a) Landmark feature pattern (Source: [Baczyk et al., 2003]).



(b) Landmark feature pattern (Source: [Jang et al., 2002]).

Figure 3.1: Different Pattern-Based Landmarks.

3.2.2 Natural Landmarks

Natural landmarks are set from real characteristics of locations. Their features are extracted from the sensory data without any changes being made in the environment. They are proper for both indoor and outdoor environments and do not need any kind of prior preparation. The identification is more complex and difficult, but the flexibility and adaptability are advantageous for mobile robot localization. One important approach to natural landmarks is:

- **Scale Invariant Feature Transform (SIFT)**
  Scale-Invariant Feature Transform (SIFT) is an approach in computer vision to detect and describe local features in images. The algorithm was invented by David Lowe [Lowe, 1999] in 1999 and has been used in mobile robotics for mapping and localization [Se et al., 2001].

  The local features are based on the appearance of the object at particular interest points and are invariant to image scaling, translation, and rotation, and partially invariant to illumination changes and affine or 3D projection. The features given by SIFT are highly distinctive, not complicated to extract, allow correct object identification with low probability of mismatch and are easy to compare against a database of local features.

3.3 Approach Implemented

For real-time navigation we rely on fast finding of features by the visual sensor. Trying to keep the detection and recognition of the landmarks as fast as possible, the approach implemented in this thesis uses solid-color artificial landmarks.

The visual system detects the landmark, recognizing its features, classifying the marker found and calculating the distance and orientation to it. This information can be feed to a navigation algorithm as a mapping capability, obstacle detection system or a localization aid mechanism.

In the next sections the steps that make up the approach are presented and explained. The steps were based on a similar approach by [Bruce et al., 2000]. In this approach it is considered that only one landmark is visible in the image at a time t.

3.3.1 Landmark Design

The design of the artificial landmark is very important since it will define the quality and difficulty of its detection and recognition. We used a cylindrical shaped object with solid color as an artificial landmark (see Figure 3.2).

The cylindrical landmarks are made out of a standard A4 colored cardboard paper and have 4 cm in radius and 21 cm in height. The proposed size makes it easy and fast to create landmarks and accomplishes the objective of being easily identifiable in an indoor environment within a considerable range. The cylindrical shape was chosen because it provides a simple visible landmark for the mobile robot having a rectangular representation by any side viewpoint from where any observation made by the mobile robot can take place.

(a) Green Landmark. (b) Blue Landmark. (c) Red Landmark.

Figure 3.2: Landmark Colors.

3.3.2 Landmark Classification

Multiple artificial landmarks can have different interpretations. For instance, a green landmark could represent an obstacle, a red landmark the goal to reach and the blue landmark a location where the mobile robot is obliged to pass. One important aspect that can not be forgotten is that different landmarks need to be distinctive from one another, meaning in this case that the colors need to be distinct of each other, not causing uncertainty on classification or when searching for a specific color in an environment.

3.3.3 Color Segmentation

The first step for detecting the landmark consists on performing color segmentation on the captured image. Previously, the landmark color features were gathered and analyzed, providing the means of empirically producing a set of rules in the RGB color space for color detection. These rules will detect the presence or not of a landmark in an image thus providing the corresponding landmark classification.

For the green landmark, the rules present in Equation 3.1 were used. For the other landmark colors, similar rules identify their colors. R, G and B correspond to the red, green and blue color components of the RGB color space. The value X is an adjustment value that is used to augment the green color component, relatively to the red and blue ones to reduce the existence of gray color pixels.

$$\text{Green Landmark: } (G \geq 130),\ (G > R + X),\ (G > B + X) \tag{3.1}$$

The application of the color segmentation process transforms the captured image into a black and white image as can be seen in Figure 3.3. White color pixels indicate the presence of the green range color and black pixels the absence of it.

3.3.4 Image Noise Reduction

As can be seen in Figure 3.3(b) some image noise may be present because of the color segmentation process. The noise present may compromise better results in future steps and therefore needs to be removed or at least reduced. The solution relies on the application of an image noise removal/reduction filter.

The filter implemented uses a 3×3 scanning window, that analyzes all the white pixels present in the image. The window checks if the pixels surrounding the current scanned pixel are mostly white or black. If they are

(a) Image captured from the visual sensor.

(b) Detection of green color pixels in the image from Figure 3.3(a) according to the rules in Equation 3.1.

Figure 3.3: Green Landmark Color Segmentation.

are mostly black ($\geq 50\%$) then the pixel is most likely noise and is erased. The result of the application of the filter implemented can be seen in Figure 3.4. Notice that the salt and pepper noise is no longer present, but the landmark still is present and continues correctly identified.

Figure 3.4: Image Noise Reduction Filter applied to the image present in Figure 3.3(b).

3.3.5 Minimum Bounding Rectangle

For more accurate calculations of distance and orientation, a rectangular boundary is created. The boundary consists on the smallest size rectangle that contains the shape detected. This boundary is used to help cope with some small variations in the shape's perspective, that can vary according to the view that the images were captured. In Figure 3.5 the bounding box is visible in red. The center is also marked with the same color.

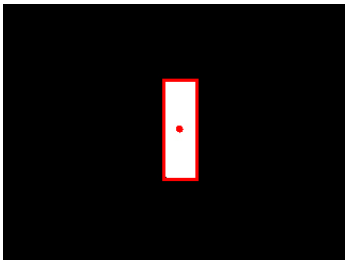


Figure 3.5: Minimum Bounding Rectangle for the landmark in Figure 3.4.

3.3.6 Distance and Orientation

For the calculation of the distance (d) and orientation (θ) from the visual sensor to the landmark we based our equations on the one of the proposed methods in [Yoon et al., 2001]. By knowing the width, height and center point of the rectangle and having already done measurements for camera calibration, the distance and orientation information can be inferred from the length in the image.

The distance between the visual sensor and the landmark is inversely proportional to the image's landmark length. For more correct distance calculation we use both the width and height of the image. They are computed independently and the final distance is given by the average of the two. The equations used are present in 3.2 .

$$d_y = k_y \times \frac{1}{y'} \qquad d_x = k_x \times \frac{1}{x'} \qquad d = \frac{d_x + d_y}{2} \tag{3.2}$$

For the angle orientation of the landmark we used the Equation 3.3. The orientation of the landmark regarding the visual sensor is computed using the x position of the center point calculated for the landmark. The x coordinate is used to calculate the angle of the landmark position in the captured image using m and l which are constants derived from the camera calibration.

$$\theta = m \times x_{LandmarkCenter} + l \tag{3.3}$$

3.3.7 Mapping algorithm

In Figure 3.6 we present the pseudocode for a simple mapping algorithm that uses visual landmark recognition as a sensor. The mobile robot has to explore all map positions in order to try to acquire the map for the whole environment. At each location, the mobile robot needs to capture an image, perform the landmark recognition and build the map with the new information obtained.

Name : Mapping Algorithm

Output: Environment Map

```
1 for all possible positions do
2 |   image = CaptureImage ();
3 |   observation = LandmarkRecognition (image);
4 |   BuildMap (observation);
5 end
```

Figure 3.6: Pseudocode for the Mapping algorithm.

3.4 Experimental Results

In order to test the implemented approach, several images were captured at predetermined distances. All images are scaled to a resolution of 320×240 pixels.

It is important to note that the distance and orientation from the visual sensor to the landmark are largely influenced by the color segmentation process. The first step of color segmentation is the most important step

in the approach described in earlier sections because all the other steps are entirely influenced by the quality of the color detection. The RGB rules have to be empirically corrected to achieve the best color detection. With a good color detection and segmentation, good results can be achieved. If the color detection and segmentation is insufficient by not having detected all of the landmark's area, then the error present in the distance and orientation calculation will be high.

Experimental results from distance and orientation calculation are available on Table 3.1 and 3.2. Absolute and relative errors are also presented. The relative error was calculated using Equation 3.4.

$$RelativeError(\%) = \frac{|Measurement - RealPosition|}{RealPosition} \times 100 \tag{3.4}$$

Using a single captured image and considering a good landmark detection and color segmentation process, the distance calculation revealed quite accurate. The average absolute error was 3.6 cm and the average relative error was 5.715%. Considering the angle orientation measurement, it revealed also quite accurate with an average absolute error of 2.11° and average relative error of 10.06%.

Landmark Position (cm)	Distance Measured (cm)	Absolute Error (cm)	Relative Error (%)
40	45.6	5.6	14
50	50.2	0.2	0.4
80	76.9	3.1	3.88
120	114.5	5.5	4.58

Table 3.1: Experimental results for the calculation of the distance to a landmark.

Landmark Position (°)	Orientation Measured (θ°)	Absolute Error (°)	Relative Error (%)
0	2.53	2.53	-
11	10.27	0.73	6.64
22	21.80	0.20	0.91
-22	-17.02	4.98	22.64

Table 3.2: Experimental results for the calculation of the angle orientation to a landmark.

Due to the design of the landmarks and the usage of a single camera, the preliminary step of calibration is necessary. This calibration has to be done for the extraction of distance and orientation from an image. Since this is yet another empirical method, it has to be carefully done to reduce possible errors. Several measurements have to be made for adjustments. The usage of only one image makes it more difficult to have extremely accurate results, because although mathematical geometry is used, the process is still very dependent on camera calibration.

Considering the mobile robot's motion, although low error values are present in the calculation of both distance and orientation, in the robot's movement these values will be constantly corrected. The mobile robot will capture images periodically and even if some measurements have some error, following measurements will adjust its perception of the environment.

3.5 Summary

Landmark recognition, although not an easy task, is a very important topic for mobile robots that need to accomplish self-localization, mapping and navigation. According to the role of the mobile robot, many different approaches can be used for landmark detection each having advantages and disadvantages associated.

For this thesis, we were interested in a simple landmark recognition approach that could be executed in real-time and with low memory usage that could be easily ported to embedded systems (e.g., smartphones). The single solid color landmarks used revealed relatively simple to detect and extract knowledge from. The approach is based on a set of steps that start with a captured image from the visual sensor. The process then continues by detecting the color of a specific landmark. Afterwards, the process evolves with image noise reduction that enables a more correct shape recognition for distance and orientation calculations. At the end of the steps, the landmark captured in an image is recognized, classified and the distance and orientation to the visual sensor is returned.

Chapter 4

Localization using Particle Filter

"Divide and Conquer."
Julius Caesar

4.1 Introduction

This chapter addresses the Particle Filter [Gordon et al., 1993], a method that aims at solving the mobile robot localization problem. The localization problem was introduced in this thesis on Chapter 2 and it represents the process of estimating where the mobile robot is, relatively to some model of the environment, using sensor measurements retrieved from that same surrounding environment.

The localization problem is a very important issue when dealing with truly autonomous mobile robots. Knowing its position in the environment enables better awareness and more precise navigation since global knowledge of the position is known. To solve this problem, this chapter uses a probabilistic approach known as Particle Filter [Gordon et al., 1993]. Other solutions for the localization problem could have been used, such as Kalman Filters [Kalman, 1960] or Markov Localization [Fox, 1998]. Our choice on Particle Filters is due mainly because of its relative conceptual simplicity and the good results it accomplishes (see [Dellaert et al., 1999], [Rekleitis et al., 2003] and [Hu et al., 2004]).

Here, our main goal is the adaptation of the Particle Filter method to a smartphone device, taking into account all of the known limitations of these devices and the nature of the method. Experimental results were conducted in order to evaluate the performance of the implemented method and to draw conclusions regarding the use of Particle Filters on smartphones.

Before the more detailed sections of this chapter, we present and define some concepts that are crucial in understanding both the Localization problem and the Particle Filter method:

- **Estimation Theory**
  Estimation Theory consists on a set of hypotheses that are maintained by a mobile robot. These hypotheses consist on a belief estimate of the robot's and surrounding obstacles position. Estimates are updated during the robot's movement throughout an environment thanks to the observations and measurements it takes.

- **Dead Reckoning/Odometry Estimation**
  Dead Reckoning is the process of modeling and estimating the robot's movement, considering only its position and orientation. This estimation model is only updated through internal odometry calculations of velocity, acceleration and time. In mobile robotics, motion models can not be based only in Dead Reckoning because of the errors from odometry estimation (wheel-slippage, unequal wheel diameters, misalignment of wheels or surface resistance).

- **Probability Density Functions (PDF's)**
  The Probability Density Function represents a probability distribution of a variable of interest based on all available information on that variable. The PDF embodies all available statistical information that can produce state estimates of that variable with a determined measure of accuracy [Gordon et al., 1993].

- **Robot Pose**
  The robot pose (see Equation 4.1) at a time t is denoted as r^t is defined as a 2-dimensional position (x, y) and an orientation (θ).

$$r^t = [x^t, y^t, \theta^t] \tag{4.1}$$

4.2 Localization

Localization can be decomposed into three slightly different problems which differ on the robot's knowledge of the environment both initially and during its movement. The three problems are presented in increasing order of their difficulty [Thrun et al., 2005]:

- **Local Position Tracking**
  Addresses local uncertainty of the robot's position whose initial pose is known. The local position will then be tracked over time.

- **Global Localization**
  Consists on the ability to determine the robot's unknown position in an *apriori* known map.

- **Kidnapping**
  Consists on a problem where a well-localized robot is secretly moved somewhere without being notified.

Another way to decompose Localization problems is through the fact of whether or not the localization algorithm controls the motion of the robot:

- **Passive Localization** is the process where the data acquired by the robot is not used to directly control the robot.

- **Active Localization** is the process where the robot is directly controlled during the localization with the purpose of minimizing its uncertainty.

4.3 Particle Filter Method

Particle Filter [Gordon et al., 1993] is a well known method to solve the Localization problem. It is a passive localization algorithm that can be used to solve both Local Position Tracking and Global Localization.

The method has a main goal of tracking a variable of interest as it evolves over time, typically with a non-Gaussian and potentially multi-model PDF. In Localization, the method is used to track que evolution of the robot's pose by building a sample-based representation of the entire PDF which approximately estimates the state of the tracked variable. The set of samples (S_i), known as particles, represent at each timestamp t a hypothesis of what the true state of the variable might be.

$$S_i^t = [r_j^t, w_j^t] : j = 1 \dots M \tag{4.2}$$

Each particle $[r_j^t, w_j^t]$ (with $1 \leq j \leq M$), is characterized by its position r^t ($[x^t, y^t, \theta^t]$) and has associated a weight w, representing the quality of that particle in the overall posterior function. The value M denotes the number of particles. According to the aim of implementation of the method, this number can vary, but it is often a large number (e.g., $M \geq 1000$). M can also be given by a function related to t.

The Particle Filter method is divided into three main phases: **Prediction**, **Update** and **Resample**. These three stages are executed and repeated over time in a loop, as the robot moves in its environment.

The **Prediction** phase uses a motion model to simulate the effect of an action on the set of particles and also considering the addition of error provided by a noise model. This simulation is used to predict the probability distribution of the mobile robot's pose after an action provided by the motion model.

- **Motion Model** is the model responsible for the mobile robot's movement.
- **Noise Model** is the model that considers the real robot error accumulation so that it can more accurately reflect a realistic model of the robot's motion. It is normally performed as a Gaussian model. Some common noise models are shown in Figure 4.1.

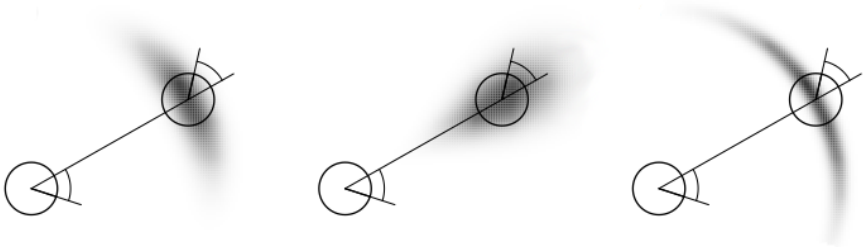


Figure 4.1: Odometry Motion Models (source: [Thrun et al., 2005]).

The **Update** phase comes into action after the prediction phase has predicted the mobile robot's movement. This phase is now responsible for sensing the environment (measurement model), gathering information that will be used to update the particles weight in order to accurately describe the moving robot's probability distribution. Without sensing the environment, the mobile robot's pose would only be estimated by dead-reckoning which would be subject to the problems presented earlier.

- **Measurement Model** is the model responsible for sensing the environment by measuring observations.

As the mobile robot is moving throughout the environment, some particles have drifted far enough for their weight to become too small to contribute to the overall PDF of the moving robot. When the number of these particles reach a certain threshold, usually a percentage of the total number of particles M, then the particle population needs to be resampled. The **Resample** phase (also known as Importance Resampling)

transforms a particle set of M particles into another particle set of the same size, thus refreshing the set by eliminating the particles with smaller weights and duplicating the ones with higher weights. If a particle has a weight under a defined threshold, then that particle has lost its relevance for calculating the robot's pose. Resampling enables the shift of focus to regions in the state space where particles have highest weights and therefore highest contribution to the robot pose estimation.

The most common methods for estimating the robot's pose are shown below. Robust Mean is usually the method that provides better results, but it is also the more computationally demanding.

- **Weighted Mean** (Equation 4.3) estimates the robot's position using a weighted mean considering all particles.

$$\overline{r}_s = \sum_{j=1}^{M} r_j^s w_j \tag{4.3}$$

- **Best Particle** (Equation 4.4) estimates the robot pose by considering the particle that at that time t has the maximum weight value.

$$\overline{r}_s = r_s^{max} \tag{4.4}$$

- **Robust Mean** (Equation 4.5) is a hybrid method which implies both the above. It corresponds to a weighted mean in a small window (ϵ) around the best particle.

$$\overline{r}_s = \sum_{j=1}^{M} r_j^s w_j : |r_s^j - r_s^{max}| \leq \epsilon \tag{4.5}$$

4.3.1 Algorithm Pseudocode

Pseudocode for the Particle Filter algorithm is presented in Figure 4.2. Lines $4-7$ are responsible for the Resample phase; lines $8-10$ are relative to the Prediction phase; lines $11-14$ are for the Update phase and lines $15-17$ are used to normalize the weights of the particles. The ESS function used in Figure 4.2 is the Effective Sample Size and can be calculated by Equation 4.6.

$$ESS_t = \frac{M}{1 + cv_t^2} \qquad cv_t^2 = \frac{var(w_t(i))}{E^2(w_t(i))} = \frac{1}{M}\sum_{i=1}^{M}(Mw(i) - 1)^2 \tag{4.6}$$

4.3.2 Problems and limitations

Particle Filters are approximate and as such are subject to approximation errors. Also, the definition of the algorithm contains some characteristics that can introduce some errors, limitations and even problems. The main issues are presented next:

- **Difficult to define the number of finite particles**
  It is difficult to adjust the number of particles to the environment, problem and objectives. It is very important to take into consideration the computational power available and the time at disposal, when choosing the number of particles to use. A low number of particles may not be enough to provide a

Name: Particle Filter Algorithm

Input : A set of Particles i at $t = 0$ $(S_i^0 = [x_j, w_j] : j = 1 \dots M)$

$W = w_j : j = 1 \dots M$;
while `Exploring` *()* **do**
  k = k + 1;
  if `ESS` *(W)* $< \beta * M$ **then**
    Index = `Resample` (W);
    $S_i^t = S_i^t(Index)$;
  end
  for *j = 1 to M* **do**
    $r_j^{t+1} = \hat{f}(r_j^t, \alpha)$;
  end
  s = `Sense` ();
  for *j = 1 to M* **do**
    $w_j^{t+1} = w_j^t * W(s, r_j^{t+1})$;
  end
  for *j = 1 to M* **do**
    $w_j^{t+1} = \frac{w_j^{t+1}}{\sum_{j=1}^{M} w_j^{t+1}}$;
  end
end

Figure 4.2: Particle Filter Algorithm (adapted from source: [Rekleitis, 2003]).

good pose estimate, while a large number of particles may in fact provide good estimation but at a high computation cost.

The best number of particles which provides a good relationship between computation cost and results can only be obtained by empirical trial of different sets of particles. The number of particles also varies due to the difficulty of different localization problems. For instance, Global Localization needs more particles than the Local Localization problem.

- **Particle deprivation**
  In a dimensional space it is possible that by random positioning no particle exists near the correct state. This can be caused by a small number of particles that cannot cover all the dimensional space available. Nevertheless this problem can happen even with a large population of particles, but with smaller probability. The problem here is the random positioning of the particles. One possible solution to minimize this problem, which is provided in [Thrun et al., 2005], consists in adding a small amount of random generated particles into the set after the resampling process.

- **Degeneracy and loss of diversity**
  Resampling too often will most certainly reduce particle diversity, since resampling will tend to concentrate particles. The Resample process has to be done only when a considerable size of the particle population does not contribute to the overall robot's pose estimation.

- **Updating the weights correctly is crucial**
  The weighting function will be directly responsible for particle quality and therefore will also be responsible for the behavior of the resampling phase and for the robot's pose estimation.

4.4 Method Implementation

The Particle Filter method implemented in this thesis is based on the approach presented in [Rekleitis, 2003]. The following sections will explain with more detail the implementation choices taken for each step of the method. This implementation addressed component modularity, thus providing easy ways to substitute any method, phase or model.

4.4.1 Environment Representation

We consider the environment to be represented as an occupancy grid map, where each grid cell matches an area of the real environment. Each cell can be free (white cell) or be occupied by an obstacle (black cell). The mobile robot is represented as the green cell with the letter "R". The goal, when needed, will be represented by a red cell with the letter "G". Figure 4.3 gives a graphical representation of a grid map example of dimensions 11×7 with obstacles, robot and goal.

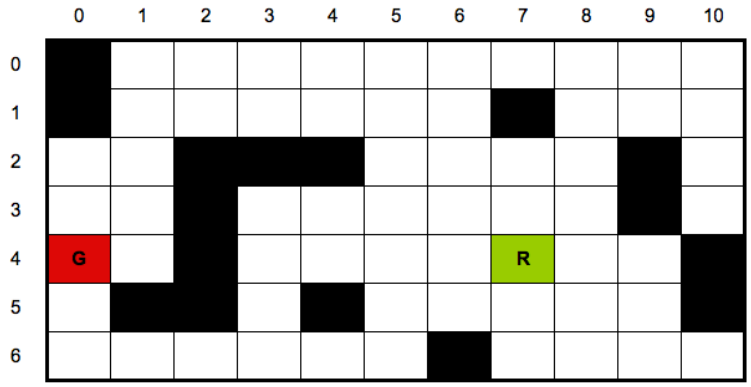


Figure 4.3: Environment represented has an occupancy grid map.

Occupancy grid representations are very useful for Particle Filter methods since they consist on a 2D representation of the real physical environment where each grid cell can be assigned with estimation probabilities of the mobile robot's position or with a reference to the presence of an obstacle.

4.4.2 Motion Model

The motion model is the robot's path planner, which is responsible for estimating at each step a path for the mobile robot's movement.

In order to provide the mobile robot with an appropriate movement so that the Particle Filter method could be properly tested and executed, two motion models were developed: an explorer type motion model and a point-to-point motion model.

- **Point-to-point** movement, is a model for robot motion that defines a concrete path that acts on the robot so it can travel from a point A in the environment to a point B.
- The **explorer** movement assumes that the mobile robot has complete knowledge of its surroundings and therefore can estimate movement directions. These directions will enable the robot to visit all possible

free positions on the current map in which it is present while also avoiding any existing obstacles. Visited positions are tagged so the robot can at any time know which are the map positions he has visited and how many it still needs to visit in order to have fully explored the environment.

Figure 4.4 illustrates the movement of a mobile robot when executing the explorer motion model within an environment. The numbers in the grid cells indicate the order in which the mobile robot visited each cell.

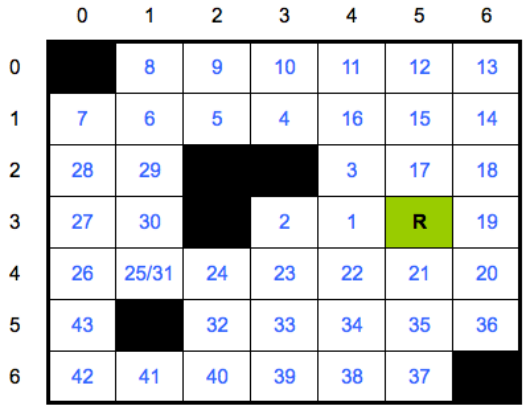


Figure 4.4: Explorer Motion Model example.

4.4.3 Noise model

Odometry noise was added to the robot's motion, modeled as a gaussian distribution (see the middle noise model from Figure 4.1), based on the noise model provided in [Rekleitis, 2003]. The possible odometry error considered for the noise was divided into rotation error and translation error. Both were experimentally established from the real odometry errors from the robotics kit used. Translation and rotation with noise is accomplished using a pseudo-random value, drawn as a sample from the guassian distribution.

4.4.4 Measurement Model

The measurement model will provide on each measurement necessary information for the **weighting function** which will update the particle's weights. In this implementation the particle's weight is considered to be a numeric value w greater than 0.

A measurement consists on an observation from the environment. This observation can be accomplished by using a single straight observation from the information present in the internal map representation or by using the visual landmark recognition method presented in Chapter 3.

4.4.5 Resampling

Resampling occurs when a considerable amount of particles within the particle population has weight values below a threshold and therefore has low contribution on the overall estimate of the robot's pose.

The resampling process recognizes particles with small weight values ($< threshold$) and replaces them with a random higher weighted particle, whose weight value is higher then the resampling threshold ($\geq threshold$). This random replacement minimizes the problem of loss of diversity. One could also consider the replacement with the current best particle, but results would not be as good since many equal particles would be created, increasing duplicates, and thus losing diversity.

When all particles have weights bellow the *threshold* then a new random set of particles is generated.

4.4.6 Robot Pose Estimate

In order to estimate the position of the mobile robot at a determined time t, we chose the best particle approach, which is the particle with the maximum weight value within the current particle set.

4.5 Experimental Results

Consider the map in Figure 4.5. It represents a room (which consists of 50 × 40 cells) with divisions in which the mobile robot starts from the green location, follows the route given in orange and stops at the red location. Table 4.1, 4.2 and 4.3 show results from the execution of the Particle Filter in this simulated environment. The results are an average for time, memory and correct pose estimate, considering 100 executions. The simulations executed in the PC environment ran on an AMD Athlon 64 X2 Dual Core Processor at 2.20 GHz with 1GB of RAM and running Windows XP SP3.

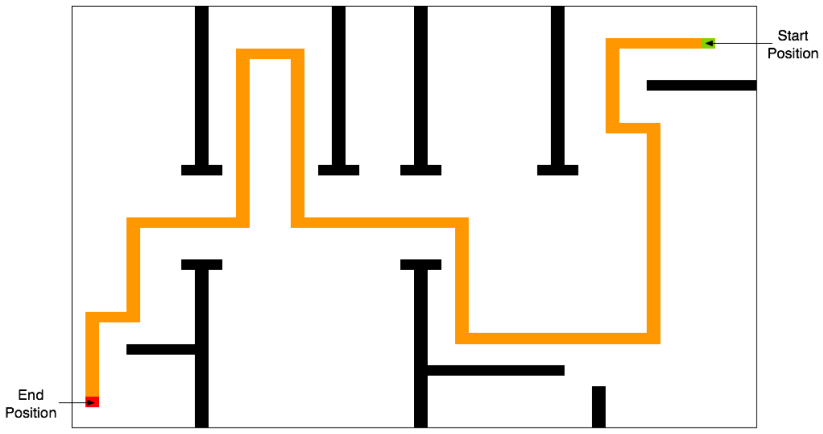


Figure 4.5: Example Map.

A correct pose estimate is assumed when the best particle is located within an area of the robot's real location, in order to accommodate errors. It is calculated using the Euclidean Distance equation for 2D points (see Equation 4.7). Here, we consider a correct pose estimate when the euclidean distance is below 10, a threshold value empirically calculated.

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{4.7}$$

As expected, time and memory consumption increases as the number of particles also increases. Also, as expected, the PC has the fastest execution times, but also the highest use of memory. Between the two

Number of Particles	Time Average (ms)	Memory Average ($bytes$)	Correct Pose Estimate (%)
10	11.71	18676.64	7 %
100	15.15	18862,72	10 %
1000	57.19	110485.60	14 %
10000	471.88	458559.60	12 %

Table 4.1: Results from the execution on the PC.

Number of Particles	Time Average (ms)	Memory Average ($bytes$)	Correct Pose Estimate (%)
10	125,88	5434,00	8 %
100	478,36	11386,91	11%
1000	4201,56	80021,00	16 %
10000	41087,60	363681,60	20 %

Table 4.2: Results from execution on Nokia N80 device.

Number of Particles	Time Average (ms)	Memory Average ($bytes$)	Correct Pose Estimate (%)
10	70,40	6094,40	5 %
100	231,58	11461,33	12 %
1000	2279,53	76249,07	15 %
10000	20636,00	379892,00	12 %

Table 4.3: Results from execution on Nokia N95 device.

smartphones, Nokia N95 has the fastest execution times and relatively the same amount of memory use as Nokia N80. These execution time results are as expected since Nokia's N95 model is equipped with a faster processor than the model N80. The correct pose estimate tends to increase along with the number of particles used in the execution, but it is not guaranteed because of the random initialization of the particles. As we can see in both the PC and Nokia N95 executions, the correct pose estimate percentage for 10000 particles is inferior to the percentage for 1000 particles.

The random initialization of the particles, also influences the correct pose estimate percentage for the experiments. Using randomness in the particles behavior makes completely indeterministic and impossible to predict the final result for pose estimates.

The next sequence of figures presents a graphical representation of the evolution of 100 particles in the execution of the Particle Filter algorithm within the environment presented earlier in Figure 4.5. The robot's real position is represented as a green circle. All particles are represented with a blue circle, labeled with the number of particles present in the same position. For this execution, the Particle Filter ran 139 steps on 16.09 ms, giving an average of 0.116 ms per step, and performed 7 resample stages.

Figure (4.6) shows the initial position of the particles in the environment. Recall that the initial positioning of the particles is done randomly. Notice that in this case the number os particles that were placed near the real robot position is low.

Figure 4.7 shows the position of the particles after the first resample phase, where it is noticeable the concentration of the particles in locations most likely to represent the real location of the mobile robot.

Figure (4.8) shows the location of the particles when the robot has finished its predetermined movement. We

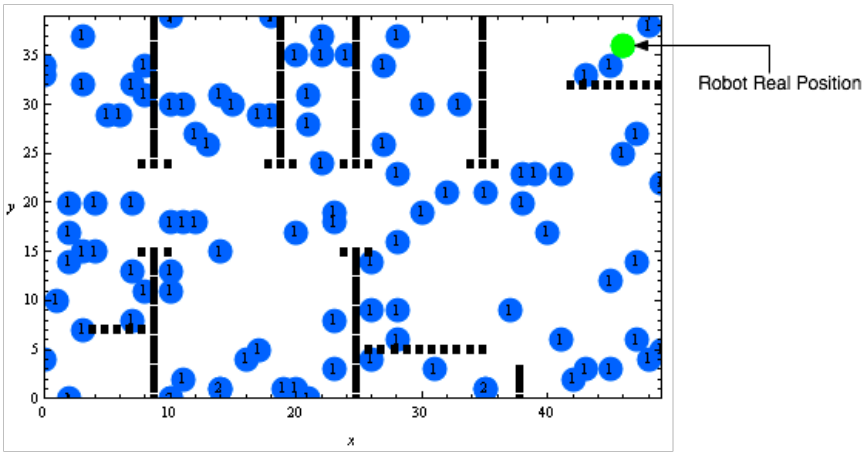


Figure 4.6: Initial particle locations.

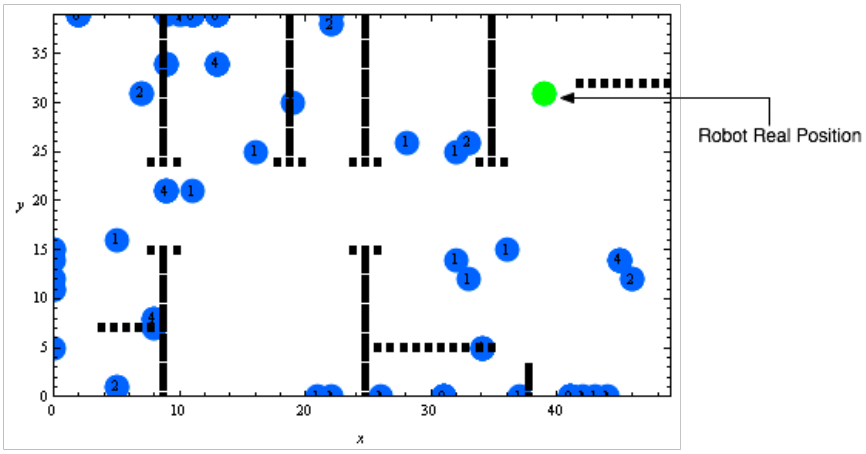


Figure 4.7: Particle locations after the first resample.

can notice that particles are again more scatter, but still, locations with higher concentration of particles are locations where according to the measurement model, are similar to the mobile robot's real position.

4.5.1 Problems and limitations identified

The results obtained from the execution of the method, mainly the low percentage of correct pose estimate and the difficult convergence of the particles to the real robot position, have highlighted two characteristics that in our view are the main constraints to achieve better results:

- The **random initialization of the particles** in the environment is an aspect that has much influence in the outcome. Considering the position and orientation in which the particles can be initialized, as well as the posterior noise that is added, executions of the same method can provide very different outcomes.
- **Sensor used for measurements** is another limitation identified. The fact that in order to simulate a vision based observation model, measurements are assumed to be forward straight observations. In this

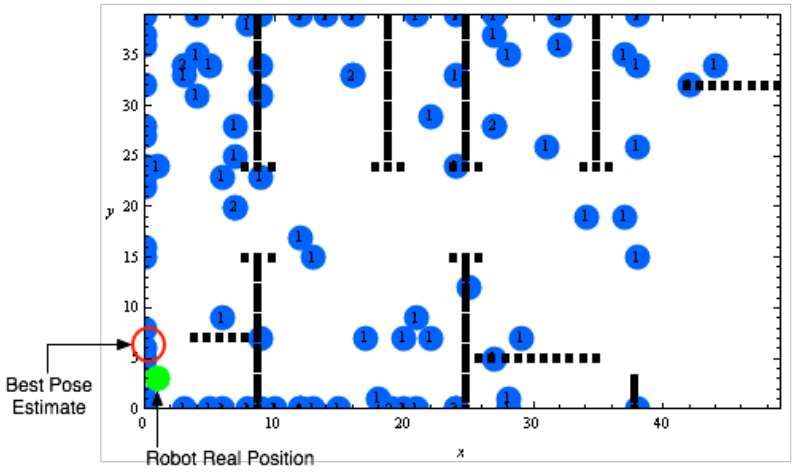


Figure 4.8: End particle locations.

type of observation the information gathered can be observed from many possible locations, therefore reducing the quality of such measurement information. Good measurements should differentiate as much as possible the particles. For example, comparing this type of measurement with a laser range sensor, which has a much broader range, straight measurements carry much less useful information in order to distinguish similar locations.

Another thing important to understand is that obstacles here are assumed to be all alike, and therefore it is not considered that one obstacle can be differentiated from another one. If obstacles could be differentiated it would be possible to distinguish among them and therefore to better distinguish similar locations.

4.6 Summary

This chapter introduces the Particle Filter method and explains the main concepts to understand the algorithm. A mobile implementation of the method is also presented. Experimental results performed on the smartphone and on the development computer show that the average execution time and memory, increases with the number of particles used. Also, increasing the number of particles tends to improve the overall pose estimate.

For localization in mobile robots, navigating at high speeds, a high number of particles, may reveal the execution of the algorithm in real-time unworkable. However, considering small amounts of particles, it is still feasible to execute such algorithm on smartphones.

Chapter 5

Path Planning using Potential Fields

"A goal without a plan is just a wish."
Antoine de Saint-Exupery

5.1 Introduction

The Potential Fields Approach [Khatib, 1986] is very used for path planning and collision avoidance due to its mathematical simplicity and elegance. It is simple to implement and can easily provide acceptable and quick results [Lee, 2004] for real-time navigation.

The objective of any path planning algorithm is to flee from obstacles and move towards a desired goal. This method is based upon the concept of attractive and repulsive forces. In potential fields, the goal is seen as a global minimum potential value, and all obstacles as high valued potential fields. The movement of the robot is then defined by the potential values present in its path, moving ideally from high to low potentials.

There are two major formulations, the **Local Potential Approach** and the **Global Potential Approach**. In this work we consider only the Local Potential Approach. For this approach, there are several different types of potential field functions, differentiating each other by the way the potential is calculated.

5.2 Global Potential Field Approach

Global Potential Field approaches were created to solve the problems inherited by the local approaches, mostly the known local minima problem. One important known global potential field method is the Navigation Function [Lee, 2004] that guarantees the existence of only one global minimum.

Some drawbacks can also be pointed to this type of approach [Lee, 2004]:

- Complete information of the environment must be know;

- Computationally expensive;
- Cannot easily adapt to changes occurring in the environment;
- Not suitable for real-time path planning.

5.3 Local Potential Field Approach

The attractive and repulsive potential fields can be calculated completely independent from each other and by adding them together we can obtain the total potential field. The scalar potential field provides the necessary local information to calculate the vector potential field force (know as action vector), direction and speed of navigation.

This approach is usually used since it does not require any prior computation before motion and it can be easily adapted to consider dynamic environments. It does not require full knowledge of the environment where the robot will navigate, permitting changes to occur. This fact is very useful, because it provides adaptability to environments with changes over time permitting real-time motion planning.

We can sum up the potential field functions as follows:

$$U_{Total}(p) = U_{Att}(p) + U_{Rep}(p) \tag{5.1}$$

$$F_{Att}(p) = -\nabla U_{Att}(p) \tag{5.2}$$

$$F_{Rep}(p) = -\nabla U_{Rep}(p) \tag{5.3}$$

$$F_{Total}(p) = F_{Att}(p) + F_{Rep}(p) = -\nabla U_{Total}(p) = -\left[\frac{\partial U}{\partial x}, \frac{\partial U}{\partial y}\right] \tag{5.4}$$

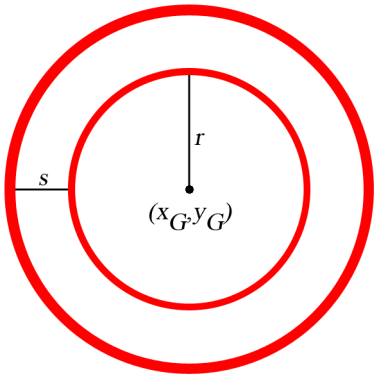
Where:

- $U_{Total}(p)$ denotes the total scalar potential field;
- $U_{Att}(p)$ denotes the attractive scalar potential field;
- $U_{Rep}(p)$ denotes the repulsive scalar potential field;
- $F_{Total}(p)$ denotes the total vector potential force which is equal to the negative gradient (∇) of the total potential field;
- $F_{Att}(p)$ denotes the attractive vector potential force;
- $F_{Rep}(p)$ denotes the repulsive vector potential force;
- p denotes the position $p = [x, y]$ of the robot.

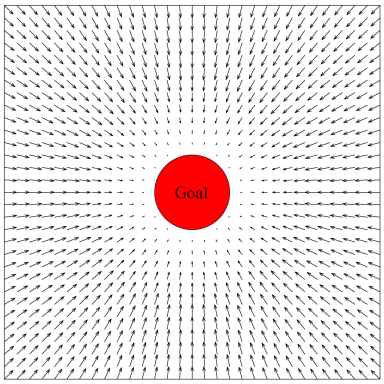
In this work, we use as basis the potential field functions presented by [Goodrich, 2002]. Other examples of potential field functions are: Khatib's FIRAS function [Khatib, 1986], Superquadratic potential function [Volpe and Khosla, 1990], and Harmonic potential function [Kim and Khosla, 1992].

5.3.1 Attractive Potential Field

The attractive potential field corresponds to the component responsible for the potentials that attract the robot towards the target goal position. At all locations in the environment the action vector will point to the target goal (see Figure 5.1(b)).



(a) Goal Representation.



(b) Attractive potential field action vectors pointing to the goal (source: [Goodrich, 2002]).

Figure 5.1: Goal representation and Attractive Potential Fields.

Usually, the action vector is found by applying a scalar potential field function to the robot's position and then calculating the gradient ($\nabla = [\nabla x, \nabla y] = \left[\frac{\partial U}{\partial x}, \frac{\partial U}{\partial y}\right]$) of that function as in Equation 5.2. In this work, we use the more direct approach of [Goodrich, 2002] where the action vector is calculated according to rules which take into account the robot's position and goal's position, dimension and area of influence. After defining:

- $[x_G, y_G]$ as the position of the goal;
- r as the radius of the goal;
- $[x_R, y_R]$ as the position of the robot;
- s as the size of the goal's area of influence;
- α as the strength of the attractive field ($\alpha > 0$).

We can compute ∇x and ∇y using the following steps:

1. Find the distance d between the goal and the robot:

$$d = \sqrt{(x_G - x_R)^2 + (y_R - y_G)^2} \tag{5.5}$$

2. Find the angle θ between the robot and the goal:

$$\theta = \tan^{-1}\left(\frac{y_G - y_R}{x_G - x_R}\right) \tag{5.6}$$

3. Set ∇x and ∇y according to the rules:

$$\text{if} \quad d < r \quad \text{then} \quad \nabla x = \nabla y = 0$$

$$\text{if} \quad r \leq d \leq s + r \quad \text{then} \quad \begin{cases} \nabla x = \alpha(d - r)\cos(\theta) \\ \nabla y = \alpha(d - r)\sin(\theta) \end{cases}$$

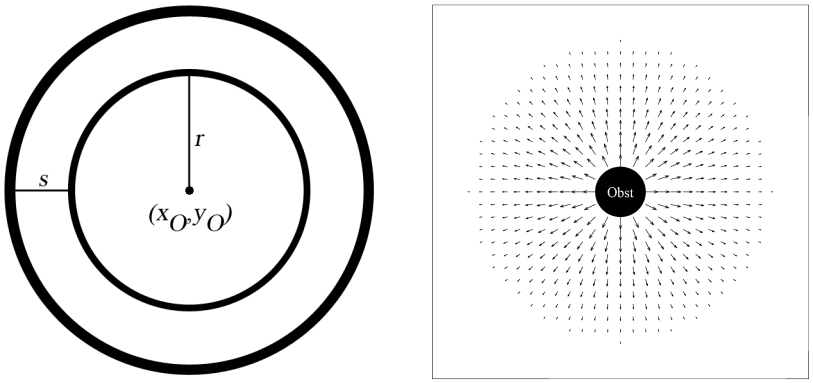
$$\text{if} \quad d > s + r \quad \text{then} \quad \begin{cases} \nabla x = \alpha s \cos(\theta) \\ \nabla y = \alpha s \sin(\theta) \end{cases}$$

The last step presents three simple rules that characterize three different behaviors for the robot according to its relative position towards the goal:

- In the first rule of step 3, $(d < r)$ means that the robot is in the goal area. In this case, no forces act and ∇x and ∇y are set to zero.
- In the second rule, $(r \leq d \leq s + r)$ means that the robot is inside the area of influence of the goal. The action vector is set using α, d and s.
- In the third and last rule, $(d > s + r)$ means that the robot is outside the goal area and also outside its area of influence. The action vector is set to with s and α thus reaching higher values.

5.3.2 Repulsive Potential Field

The repulsive potential field is the component that is responsible for forcing the robot to stay away from the obstacles it encounters on its path. In Figure 5.2(b) all repulsive action vectors point away from the obstacle surface driving the robot away from the obstacle.



(a) Obstacle Representation.

(b) Repulsive potential field action vectors pointing away from the obstacle (source: [Goodrich, 2002]).

Figure 5.2: Obstacle representation and Repulsive Potential Fields.

Similarly to the Attractive Potential, we calculate the repulsive action vector using the approach provided in [Goodrich, 2002]. After defining:

- $[x_O, y_O]$ as the position of the obstacle;
- r as the radius of the obstacle;

- $[x_R, y_R]$ as the position of the robot;
- s as the size of the obstacle's area of influence;
- β as the strength of the repulsive field ($\beta > 0$)

We can compute ∇x and ∇y using the following steps:

1. Find the distance d between the obstacle and the robot:

$$d = \sqrt{(x_O - x_R)^2 + (y_R - y_O)^2} \tag{5.7}$$

2. Find the angle θ between the robot and the obstacle:

$$\theta = \tan^{-1}\left(\frac{y_O - y_R}{x_O - x_R}\right) \tag{5.8}$$

3. Set ∇x and ∇y according to the rules:

$$\text{if} \quad d < r \quad \text{then} \quad \begin{cases} \nabla x = -sign(\cos(\theta))\infty \\ \nabla y = -sign(\sin(\theta))\infty \end{cases}$$

$$\text{if} \quad r \leq d \leq s + r \quad \text{then} \quad \begin{cases} \nabla x = -\beta(s + r - d)\cos(\theta) \\ \nabla y = -\beta(s + r - d)\sin(\theta) \end{cases}$$

$$\text{if} \quad d > s + r \quad \text{then} \quad \nabla x = \nabla y = 0$$

Similar to the attractive potential rules, these rules are also simple and characterize three different behaviors for the robot according to its position relative to the obstacle. It is important to notice that all action vectors need to point away from the obstacle, hence the need to use negative values.

- In the first rule of step 3, the robot is within the radius of the obstacle, so the action vector needs to be infinite, expressing the need to escape from the obstacle.
- In the second rule, where the robot is outside the obstacle's radius but inside its area of influence, the action vector is set to a high value in order to express the need to escape the current location.
- In the third rule, where the robot is outside the area of influence of the obstacle, the action vector is set to zero, meaning that no repulsive forces are acting on the robot.

Since the repulsive force only acts when the robot is inside the area of influence of the obstacle, the value of s must be carefully chosen. A small value for s can cause trajectory problems by causing abrupt changes on the path and some constraints on the speed of the robot. A large value for s may cause also problems on the robot's movement since it can constrain movement in small places where the robot could pass.

The repulsive force has the objective of repelling the robot only if it is close to an obstacle and its velocity points towards that obstacle.

5.3.3 Total Potential Field

Knowing both the attractive and repulsive potential, we are now interested in combining them in order to obtain the total potential field scalar value and vector force present in the environment as indicated by the equations (5.1) and (5.4).

The combination of the two fields can be done by adding them together (the combination of both potential fields can be seen in Figure 5.3). The action vector can be decomposed in its 2D components as:

$$\nabla x = \nabla_{Repulsive} x + \nabla_{Attractive} x \tag{5.9}$$

$$\nabla y = \nabla_{Repulsive} y + \nabla_{Attractive} y \tag{5.10}$$

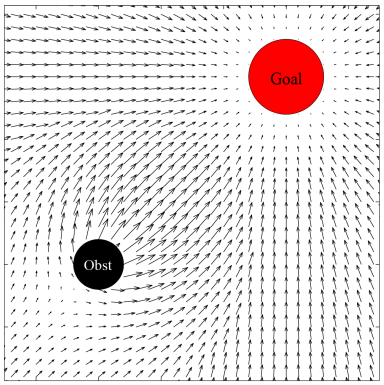


Figure 5.3: Total potential field action vectors on an environment with a goal and an obstacle (source: [Goodrich, 2002]).

With the total potential field calculated, it is possible to determine the velocity v and the angle θ of the resulting action vector. The velocity v can be calculated through $\sqrt{\nabla x^2 + \nabla y^2}$. The angle θ can be calculated using $\tan^{-1}\left(\frac{\nabla y}{\nabla x}\right)$.

5.3.4 Algorithm

Using the Potential Field Approach, motion planning is performed in an iterative fashion. At each iteration the forces are calculated by the potential functions with the current environment characteristics providing a direction of motion that is used by the robot for some position increment. For the approach presented, the pseudocode can be seen in Figure 5.4. The Potential Field functions with the calculation of the potential values can be seen in Figures 5.5, 5.6 and 5.7.

Figure 5.4 shows the main control loop for the Potential Field Approach. While not in the goal position, a directional action vector is calculated from the total potential field force. This action vector is responsible for determining the velocity and angle for the robot's direction of movement. The *UpdateRobotPosition* function takes the action vector and computes the next position of motion for the mobile robot.

Figure 5.5 shows the calculation of the total potential field force (according to the Equation 5.4) that is dependent on the attractive and repulsive components. Figure 5.6 calculates the attractive potential com-

Name : Potential Field Algorithm

Input : Robot Position (rp), Goal Position (gp), Obstacle Positions (op[])
Output: Directional Action Vector

```
1 while NotInGoalPosition do
2 |   ActionVector ← CalculateTotalPotentialForce (rp, gp, op[]) ;
3 |   DetermineVelocity (ActionVector);
4 |   DetermineAngle (ActionVector);
5 |   UpdateRobotPosition (ActionVector) ;
6 end
```

Figure 5.4: Pseudocode for the Potential Field Approach algorithm.

ponent (Equation 5.2) illustrated in the method shown in Section 5.3.1. Figure 5.7 is responsible for the repulsive component (Equation 5.3) presented in Section 5.3.2.

Name : CalculateTotalPotentialForce

Input : Robot Position (rp), Goal Position (gp), Obstacle Positions (op[])
Output: Directional Action Vector

```
1 AttractiveActionVector ← CalculateAttractivePotentialForce (rp, gp);
2 RepulsiveActionVector ← CalculateRepulsivePotentialForce (rp, op[]);
3 TotalActionVector ← AttractiveActionVector + RepulsiveActionVector;
```

Figure 5.5: Pseudocode for the *CalculateTotalPotentialForce* function.

5.4 Problems and Limitations

The Potential Fields Approach, although simple and elegant, inherits some problems and limitations that affect its performance on delivering paths for navigation. The topics where the method's problems arise are:

- Obstacle Geometric Modeling
- Concave Shaped Obstacles
- Obstacle Grouping
- Goals Non-Reachable with Obstacle Nearby (GNRON)
- Closely positioned obstacles
- Non-Optimal paths
- Local Minima

Some of the problems and limitations presented can be reduced or even solved by tuning the field strength parameter and the distance of influence parameter in the calculations of the fields. The downside here is that parameter tuning solutions might not work for all cases, bringing good results for some, but bad results for others. We outline in the next sections some solutions for each problem.

Name : CalculateAttractivePotentialForce

Input : Robot Position (rp), Goal Position (gp)
Output: Attractive Directional Action Vector (AttractiveVector)

```
1  d ← CalculateDistance (rp,gp);
2  θ ← atan((gp.Y − rp.Y), (gp.X − rp.X));
3  α ← ATTRACTIVE_FIELD_STRENGTH;
4  r ← GOAL_RADIUS;
5  s ← ATTRACTIVE_FIELD_LENGTH;
6  if (d < r) then
7  |  AttractiveVector.X ← 0;
8  |  AttractiveVector.Y ← 0;
9  end
10 else if ((d ≥ r)&&(d ≤ s + r)) then
11 |  AttractiveVector.X ← α × (d − r) × cos(θ);
12 |  AttractiveVector.Y ← α × (d − r) × sin(θ);
13 end
14 else if (d > (s + r)) then
15 |  AttractiveVector.X ← α × s × cos(θ);
16 |  AttractiveVector.Y ← α × s × sin(θ);
17 end
18 return AttractiveVector;
```

Figure 5.6: Pseudocode for the *CalculateAttractivePotentialForce* function.

Name : CalculateRepulsivePotentialForce

Input : Robot Position (rp), Obstacles Position (op[])
Output: Repulsive Directional Action Vector (AttractiveVector)

```
1  β ← REPULSIVE_FIELD_STRENGTH;
2  r ← OBSTACLE_RADIUS;
3  s ← REPULSIVE_FIELD_LENGTH;
4  forall Obstacle Position op in op[] do
5  |  d ← CalculateDistance (rp, op);
6  |  θ ← atan((op.Y − rp.Y), (op.X − rp.X));
7  |  if (d < r) then
8  |  |  RepulsiveVector.X(RepulsiveVector.X + (−1 × sign(cos(θ)) × ∞ × s));
9  |  |  RepulsiveVector.Y(RepulsiveVector.Y + (−1 × sign(sin(θ)) × ∞ × s));
10 |  end
11 |  else if ((d ≥ r)&&(d ≤ s + r)) then
12 |  |  RepulsiveVector.X(RepulsiveVector.X + (−1 × s × (s + r − d) × cos(θ));
13 |  |  RepulsiveVector.Y(RepulsiveVector.Y + (−1 × s × (s + r − d) × sin(θ));
14 |  end
15 |  else if (d > (s + r)) then
16 |  |  RepulsiveVector.X(RepulsiveVector.X + 0);
17 |  |  RepulsiveVector.Y(RepulsiveVector.Y + 0);
18 |  end
19 end
20 return RepulsiveVector;
```

Figure 5.7: Pseudocode for the *CalculateRepulsivePotentialForce* function.

5.4.1 Obstacle Geometric Modeling

In the previous calculations of the attractive and repulsive field, the distance functions consider all obstacles used to be circular, not requiring any distance calculations. However, in the real world not all obstacles are of the same size, shape and aspect, and with asymmetric obstacles the potential functions are difficult to use. Since in real life there are many different types of obstacles, it is obvious that we need to consider other obstacle shape types besides circular ones. In this work we also consider rectangular obstacles.

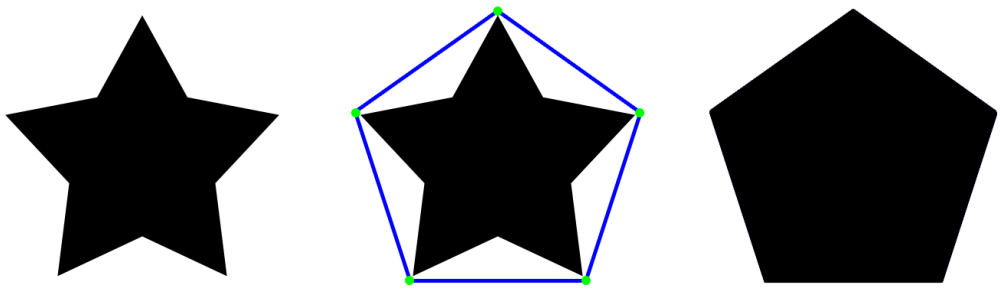
With circular shapes we only need to know the center of the circular shape and its radius. For the rectangular shaped obstacles we use the obstacle's four vertices.

As a representation, each rectangular obstacle could be seen as a set of circular ones or by the smallest circular obstacle able to contain it. Of course these representations are not very accurate and do not solve most problems. In order to effectively consider the true dimensions of the obstacle and to accurately calculate the distance of the robot to them, we need to know exactly the surface point of the obstacle nearest to the robot. In [Khatib, 1986] and [Kim and Khosla, 1992] equations for this calculation are presented.

In this work, for rectangular shaped obstacles, the closest surface point was calculated by first determining the rectangle's closest edge and then calculating the perpendicular from the robot to that edge.

5.4.2 Concave Shaped Obstacles

Concave obstacles, due to their shape, are more likely to create locations with high local minima probability which may trap the robot. By definition a polygon is concave if at least one of its internal angles is greater than 180° (this creates dents). To minimize the existence of the concave dents, it is possible to close them, creating a convex shaped obstacle that encloses the concave. This transformation can be achieved using the Convex Hull algorithm [Cormen and Rivest, 2001] as seen in Figure 5.8.



(a) Concave Obstacle.

(b) Transformation from concave to convex.

(c) Convex Obstacle.

Figure 5.8: Transformation of a concave obstacle to a convex obstacle.

Addressing this problem can help reduce the number of local minima in the environment by preventing the robot from entering problematic areas which can trap it.

5.4.3 Obstacle Grouping

One difficulty present in the robot's navigation is the presence of multiple obstacles closely positioned, since all their forces are taken into account on the calculations of the repulsive potential. One approach to solve or minimize this situation is to group these closely positioned obstacles as one bigger obstacle.

In theoretical terms, the existence of a bigger obstacle could ease the navigation, relieving the robot from moving towards an obstacle dense zone. In practice, the bigger obstacle can also compromise an optimal path towards the goal.

5.4.4 Goals Non-Reachable with Obstacle Nearby (GNRON)

Many times obstacles are not close to the goal and as the robot comes closer to the goal, the repulsive forces from the obstacles are negligible. A problem arises when there are obstacles near the position of the goal. As the robot approaches the goal, it also approaches the obstacles near the goal, being affected by both the attractive and repulsive forces. According to the rules presented earlier for the calculation of the potential fields, the repulsive force will be much stronger than the attractive force, thus making the robot move away from the goal.

The robot may later be again attracted to the goal, but the repulsive force will always be present, winning over the attractive force, making the goal position unreachable and thus not the global minimum of the total potential.

This problem is studied by [Ge and Cui, 2000]. To avoid this problem they presented new potential functions that take into consideration the relative distance between the robot and the goal.

5.4.5 Closely Positioned Obstacles

When obstacles are near each other and there is only a narrow passage between them, depending on the influence of the repulsive forces from the obstacles, the robot can pass through the passage or simply turn away [Koren and Borenstein, 1991].

In other conditions, such as corridors, where obstacle walls are positioned so that as the robot escapes from one, comes into contact with the force of another, there can occur oscillations on the robot's movement. This oscillation exists due to the necessity of the robot to escape from multiple closely positioned obstacles it comes across.

This problem can be solved by tuning the parameters of the potential field equations, since they mostly depend on the distance of influence of the forces and on their field strength.

5.4.6 Non-Optimal paths

This problem is mostly due to characteristics of the potential field approach, since it is very difficult to predict the trajectories. Using the Potential Fields does not guarantee optimal shortest paths.

In mobile robotics this problem can have serious impact because of its implications on energy consumption, which increases proportionally to the length of the path.

5.4.7 Local Minima

A local minimum location can trap the robot before reaching its goal. The local minima exists when the action vector of the attractive potential field is stronger than the action vector of the repulsive potential field and the robot is close to obstacles. It can also occur in locations where attractive and repulsive forces are in equilibrium. This situation translates itself as an action vector that does not provide a direction, causing the robot to be trapped in that location.

Detection of the Local Minima: When a robot reaches a location where it encounters a local minima, it will be trapped, being unable to continue its path to the global minimum (goal) of the environment. Our approach to determine whether a robot encounters itself at a local minimum situation consists on a very simple method that makes use of an n step lookahead to the near future movement. Before moving, the robot performs a prior calculation where it tests whether in the near future it will be trapped in a local minimum. The test is made by mathematically calculating its n next positions and verifying if its location is not changing. If after moving n positions the robot is at the same location in the environment (with some given defined position error) there has been detected a local minimum.

To treat the local minima problem two classes of solutions are presented in [Lee, 2004]:

- **The redefinition of the potential functions with no or few local minima**
  There has been some work done on redefinition of the local potential field functions in order to solve the local minima problem. The objective of these redefinitions is to minimize the existence of local minima in the functions besides their global minimum. One example are the Harmonic Potential Functions [Kim and Khosla, 1992].
- **Escape Techniques**
  The usage of efficient search techniques with the capability of providing directions and paths for escaping from local minima locations. We focus our attention on the escape techniques: **Random Escape**, **Perpendicular Escape**, **Virtual Obstacle Concept Escape** and **Simulated Annealing Escape**.

Random Escape Approach: This escape technique is the simplest approach where a random direction of the robot's possible movements is selected.

Perpendicular Vector Escape Approach: This escape technique is studied and proposed in the work [Veelaert and Bogaerts, 1999]. It uses the perpendicular vector of the robot's movement vector as the escape direction.

For any vector, we can choose from two available perpendicular directions. Considering a vector $\overrightarrow{a} = [x, y]$ we denote $\overrightarrow{a^{\perp 1}} = [-y, x]$ and $\overrightarrow{a^{\perp 2}} = [y, -x]$ as the perpendicular vectors.

Having two perpendicular directions we face a decision point: choose always only one of them, or a combination of the two. The choice of only one of the perpendicular directions can actually give a good escape path, but it is too much dependent on the environment where the robot is currently on. Using a combination of both perpendicular directions reveals inefficient escape paths as it can block the robot's movement within a determined region.

Virtual Obstacle Concept Escape Approach: This is an escape technique for the local minima problem introduced by [Park and Lee, 2003]. This technique consists in the real time creation of virtual obstacles that oblige the robot to reformulate its path. The virtual obstacle is generated when the robot encounters itself trapped in a local minima situation. Due to normal potential field planning the robot will move away from the virtual obstacle hence moving away from the local minima location.

Moreover, besides providing the robot with an escape to the local minimum, the virtual obstacle also prohibits the robot from going back to that location.

Simulated Annealing Escape Approach: This escape consists on an algorithm for iterative improvement and optimization in the presence of a large search space [Kirkpatrick et al., 1983].

As an escape technique, Simulated Annealing was used by [Park et al., 2001] to provide a solution to the local minima problem. Figure 5.9 presents the Simulated Annealing Escape approach.

Name: Simulated Annealing Algorithm

```
1  Set P = S;
2  Set T = T_0;
3  while T ≥ T_f and NotEscaped do
4  |  Pick random neighbor P' of P;
5  |  Calculate U(P') which is the potential value at P;
6  |  Set △ = U(P') − U(P);
7  |  if △ ≤ 0 then
8  |  |  set P = P'
9  |  end
10 |  if △ ≥ 0 then
11 |  |  set P = P' with probability e^{−△/T}
12 |  end
13 |  if U(P') ≤ U(S) then
14 |  |  successful escape
15 |  end
16 end
17 if NotEscaped then
18 |  return failure
19 else
20 |  escape
21 end
```

Figure 5.9: Simulated Annealing Algorithm applied to Potential Fields.

5.5 Implementation

As referred before, the robot is represented as a circle shaped object with a certain radius and there can only be one in the environment. The target goal is a circle shaped object with a certain radius and length of area of influence. The obstacles can be either circular shaped or rectangular shaped objects, with any given size and length of area of influence.

For the environment representation, in order to save memory and represent only the information needed, an environment is represented by its size, width and length and by the xy coordinates of the entities present in

the environment.

The Potential Field method receives as input the environment representation and provides the robot with information for it to navigate from its start position to the goal. At each position in the robot's path, all forces are calculated and the robot's motion is corrected. This fact allows for dynamic goals and obstacles.

The pseudocode presented in Figure 5.10 shows the main sequences of control for the potential field method discussed in this chapter with local minima detection and escape techniques. Further parts of the algorithm can be seen in Figure 5.11, 5.12 and 5.13.

Name: Potential Field Approach Algorithm

Input: Robot Position (rp), Goal Position (gp), Obstacle Positions (op[])

```
1 while NotInGoalPosition do
2 |  if IsInLocalMinima(rp, gp, op[]) then
3 |  |  Escape(rp, gp, op[]);
4 |  else
5 |  |  Move(rp, gp, op[]);
6 |  end
7 end
```

Figure 5.10: Main control sequence for the Potential Field Approach with local minima detection and escape techniques.

Considering the navigation of the robot from its initial position to the goal position, the algorithm needs to provide the path until the robot reaches its goal position. While not in the goal position, on the robot's movement, local minima situations need to be detected (see Figure 5.11) for the robot to escape the location and continue with its movement.

Name : IsInLocalMinima

Input : Robot Position (rp), Goal Position (gp), Obstacle Positions (op[])
Output: True if local minimum location detected, false otherwise

```
1 LookahedPosition ← MovementLookahead(NumberOPositions, rp, gp, op[]);
2 if LookaheadPosition == InitialPosition ± PositionError then
3 |  return true
4 else
5 |  return false
6 end
```

Figure 5.11: Detection of local minima.

For detecting local minima locations, a lookahead function is used where n movements are calculated and compared with the current robot position. If after n movements the robot is still in the same position, taking into account a certain position error, then the robot is considered to be in a local minimum location.

Upon a local minima location, the escape solution consists in providing a direction vector from where the robot may escape the current position according to different possible escape techniques (see Figure 5.12).

The movement of the robot is provided by the potential field function, where each force gives the strength, velocity and angle of movement. For more information on these functions refer to Figures 5.5, 5.6 and 5.7.

Name : Escape

Input : Robot Position (rp), Goal Position (gp), Obstacle Positions (op[])
Output: Escape Directional Vector

```
1 EscapeDirection ← EscapeTechnique(rp, gp, op[]);
2 return EscapeDirection
```

Figure 5.12: Escapes a local minimum location.

Name : Move

Input : Robot Position (rp), Goal Position (gp), Obstacle Positions (op[])
Output: Directional Action Vector for robot movement

```
1 ActionVector ← CalculateTotalPotential (rp, gp, op[]) ;
2 DetermineVelocity (ActionVector);
3 DetermineAngle (ActionVector);
4 UpdateRobotPosition (ActionVector) ;
```

Figure 5.13: Provides the path for the movement of the robot.

5.6 Experimental Results

The experimental testing conditions were based upon a set of scenarios to evaluate the algorithms. The experimental information presented next was gathered throughout simulations run on a Apple MacBook with an Intel Core 2 Duo 2.0GHz processor, 2GB of RAM memory and running OSX Leopard.

In order to evaluate the performance of each version of the potential field algorithm, it is necessary to use a set of metrics that represent a measure of the quality of the path calculated:

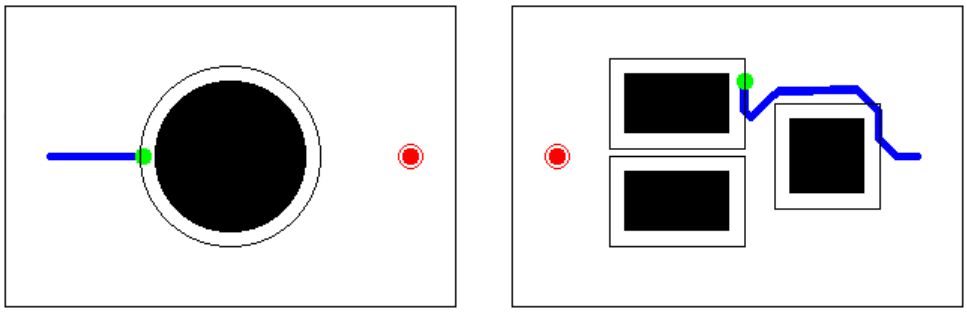
- **Time Metric:** This metric consists of measuring the time that passed from the initial robot movement until it reaches the goal.
- **Space Metric:** This metric counts the number of positions the robot visited from the start position until the target goal position is reached.

In the following examples (see Figure 5.14, 5.15 and 5.16) it is illustrated the value of the escape techniques, mainly the Virtual Obstacle Escape and the Simulated Annealing Escape. The black shapes represent obstacles, the green circle represents the robot, the red circle represents the target goal and the blue line represents the path taken by the robot's movement.

The Random Escape and Perpendicular Vector Escape where not included here since evaluation showed them not satisfactory. They were not real-time efficient in evading local minima locations.

In Figures 5.14(a) and 5.14(b) the execution of the potential field method neither implements the detection of local minima nor escape techniques, thus, in the provided scenarios it falls in local minima situations. Figures 5.15 and 5.16 show the same scenarios but having the potential field method implement detection of local minima and using escape techniques.

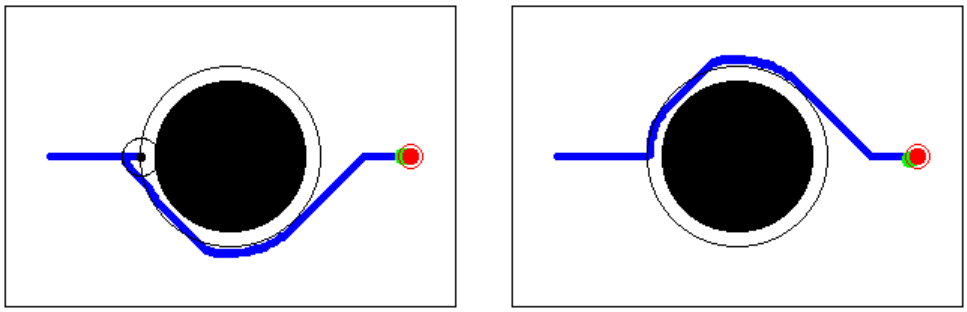
In both 5.15 and 5.16, the local minimum location is escaped and the path to the goal can be completed. In the first case, the results are very similar both in time and space metrics. This result is comprehensible since both escape solutions provide similar paths and the simulated annealing algorithm is more complex than the



(a) Local minima detection (Time Metric = ∞; Space Metric = ∞).

(b) Local minima detection (Time Metric = ∞; Space Metric = ∞).

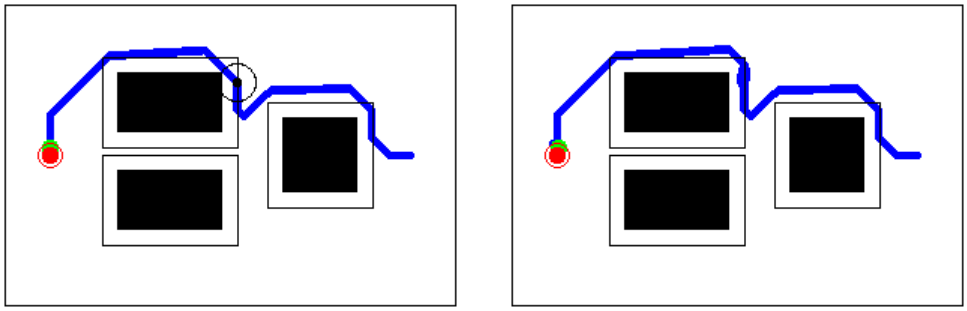
Figure 5.14: Potential Fields navigation path encounters local minima locations.



(a) Virtual Obstacle escape technique (Time Metric = 20ms; Space Metric = 264pos).

(b) Simulated Annealing escape technique (Time Metric = 37ms; Space Metric = 262pos).

Figure 5.15: Escape techniques applied to 5.14(a).



(a) Virtual Obstacle escape technique (Time Metric = 20ms; Space Metric = 264pos).

(b) Simulated Annealing escape technique (Time Metric = 940ms; Space Metric = 396pos).

Figure 5.16: Escape techniques applied to 5.14(b).

virtual obstacle concept. In the second case, simulated annealing takes more time and space to reach the target goal, performing worse than the virtual obstacle escape. The path although very similar took more positions to escape the local minimum location.

Simulated Annealing and Virtual Obstacle Concept are two valid solutions to escape local minima locations. Their performance varies due to characteristics of the environment, but in most cases Virtual Obstacle Concept is faster and provides shorter paths. This is because of its low complexity and to the fact that it

takes advantage from the mathematical characteristics of the repulsive potential field, where in the presence of obstacles provides a quick directional escape path.

Figure 5.17 shows a more complex environment that represents a building floor. The approach provides a clear path from the initial position to the goal position avoiding all obstacles in its path.

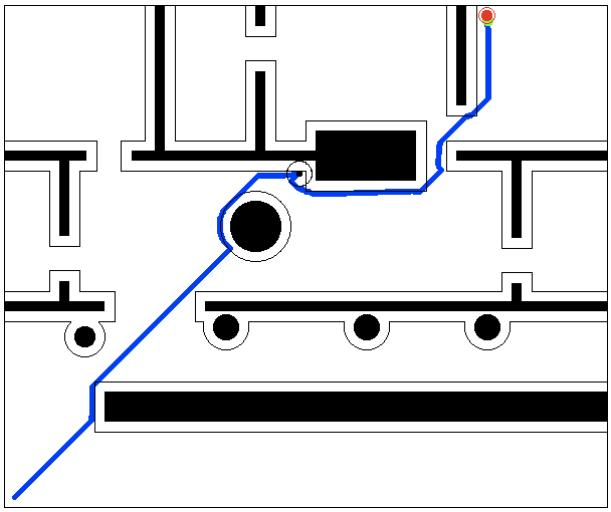


Figure 5.17: Potential Fields with Virtual Obstacle Escape on a complex environment (Time Metric = 7891 ms; Space Metric = 673 pos).

From the experimental results performed, it was possible to observe that in the cases where the robot must go back on its path to correct its motion, the Potential Fields approach does not perform very well. For example, in a labyrinth environment, the approach would have enormous difficulties finding a path, falling constantly in local minima locations, eventually becoming completely trapped, unable to continue its motion.

From the analysis of the method, we can conclude that many advantages are given from the Potential Field Approach. In simple straightly forward environments, the method actually performs really well, but in environments more complex where sometimes it is needed to go back in a path to correct the movement, Potential Fields may fail. Potential Field Approach can have a very important role as a secondary method in path planning. Other path planning algorithms can use Potential Fields as a solution to aid on specific obstacle avoidance.

5.7 Summary

In this chapter, a local potential field approach for path planning is described, analyzed and discussed. One of the most important advantage of the local potential field method is its easy adaptation to real time environments, since it is possible to have goal and obstacle movements. One verified disadvantage is that no solution adapts perfectly to all environment scenarios. The main problem identified is the Local Minima for which are seen escape techniques that are exemplified and tested. A lookahead method for detection of local minima was used. Although more computations must be done, local minima situations are detected before trapping the robot, therefore preventing the robot from describing oscillations on its movement when in the presence of these locations.

Chapter 6

Embedded System Prototype

"Actions lie louder than words."
Carolyn Wells

6.1 Introduction

This chapter discusses the embedded system proposed in this thesis, represented by Figure 1.3. Here we describe the embedded system prototype, justifying the choice of components and their role in the overall system. As mentioned before, the mobile robot used is the Lego Mindstorms NXT kit, and for the smartphone we used the Nokia N80 and Nokia N95 models.

The system set for this thesis is composed by the mobile robot and the smartphone which communicate using bluetooth technology. Together, they represent a robotic system that is remotely controlled by a mobile phone which guarantees the execution of the navigation algorithms in real-time. With the intention of abstracting bluetooth communication issues and robot control, we developed a middleware component. The middleware therefore guarantees better interoperability between the two system components.

6.2 Mobile Robot

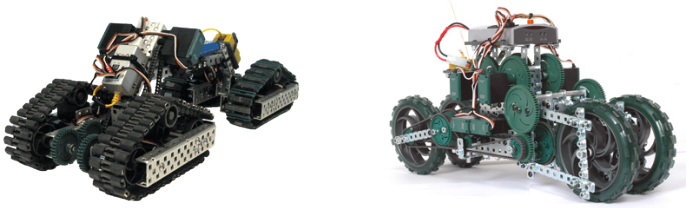
The mobile robot is the component of the overall system which provides the physical testing platform for the developed navigation algorithms. This platform provides a real physical testing facility which enables better perception of the algorithm's behavior, as well as faster error and problem detection, when compared with all initial testing done on simulations on a standard computer environment.

Since the main aim of this thesis did not involve any complex hardware work, the mobile robot component should be easy to use, to develop for and to test. Bearing these requirements in mind, two mobile robotic kits were considered:

- Lego Mindstorms NXT Kit [Lego, 2008b] (See Figure 6.1)
- Vex Labs Robotics Kit [VEXRobotics, 2008] (See Figure 6.2)

(a) NXT Brick and sensors (b) NXT Simple Robot Model (c) NXT Scorpion Robot Model

Figure 6.1: Lego Mindstorms NXT.



(a) Vex Tomahawk Robot (b) Vex Magbot M3 Robot

Figure 6.2: Vex Robot Examples.

Due to its simplicity and functionality out-of-the-box, namely the already embedded bluetooth interface for communication, the kit by Lego was chosen as our mobile robot platform.

6.2.1 Lego Mindstorms NXT

The mobile robotic kit Lego Mindstorms NXT [Lego, 2008b] is a programmable robotics kit released by the well known Lego Group in 2006, being today one of the simplest, easy to use and cheapest mobile robotic kits. It is easy to setup and can be adjusted to the programmers developing needs as can be seen in Figure 6.1.

The main component in the kit is a rectangular-shaped computer, called the NXT Brick which can be seen in the center of Figure 6.1(a), along a number of available sensors and motors. It includes a greyscale LCD display, four buttons and a built-in sound speaker. The NXT Brick is the central processing unit of the mobile robot, being responsible for all I/O communication (wired or bluetooth) and for providing input from up to four sensors and output of up to three motors. For a detailed technical hardware specification, see Table B.1.

Programming for the NXT Brick

Programming for the NXT Brick can be done using different approaches, each one using a different programming language and therefore having specific objectives, advantages and disadvantages:

- NXT-G [Lego, 2008a]
- Next Byte Codes (NBC) & Not eXactly C (NXC) [Hansen, 2008a]

- Not Quite C (NQC) [Hansen, 2008b]
- leJOS NXJ [Solórzano et al., 2008a]
- nxtOSEK [Chikamasa, 2008]
- MATLAB and Simulink [MathWorks, 2008]

For this thesis we used the leJOS NXJ [Solórzano et al., 2008a] which is based on the Java programming language and requires the use of a custom firmware on the NXT Brick. The choice for this approach was due to the intended development in Java and due to the leJOS NXJ recognized overall quality, development support and available resources.

leJOS NXJ

leJOS NXJ [Solórzano et al., 2008a] is a firmware for the Lego's NXT Brick which includes a tiny Java Virtual Machine (JVM). It can be uploaded to the Brick offering a good API for anyone developing applications that use the Lego's NXT Mindstorms kit.

The leJOS also provides a Java programming language environment [Bagnall, 2007], with a great number of class libraries that support the most common high level Java functions in conjunction with methods to access most of the capabilities, sensors and funcionalities of the NXT.

The leJOS NXJ project was developed by the leJOS development team (José Solórzano, Brian Bagnall, Jürgen Stuber and Paul Andrews) and is available since 2006.

6.3 Smartphone

Within the purpose of this thesis, the smartphone component assumes the role of the main processing unit. It is responsible for the execution of all navigation algorithms as well as communicating with the mobile robot. Other auxiliary tasks such as image capture using the built-in camera, user input and data storage are also of the smartphone's responsibility.

The use of a smartphone guarantees a small embedded system, that can be easily programmed and capable of processing complex navigation algorithms. Today, smartphones are very important in the mobile devices industry being recognized as good computational devices thanks to their functional capacities (e.g., wireless communications, built-in photo and video camera, low energy consumption).

6.3.1 Smartphone Programming

In terms of development, the applications created for the smartphone platform were programmed using the Java programming language in its Micro Edition (J2ME), specific for embedded devices. The main reasons for programming in J2ME were:

- **Portability across a wide range of devices**, since Java is supported by the great majority of commercial smartphones;

- **Easy development**, thanks to powerful API libraries that support the Java J2ME development which are also available for J2SE development;
- Version of a well known and stable **programming language** that throughout the years already proved its value;
- Many **resources available** for J2ME developers;

Figure 6.3 presents an overview of the components of J2ME technology and how it is related to the other available Java Technologies. More information about J2ME can be found in [SunMicrosystems, 2008a].

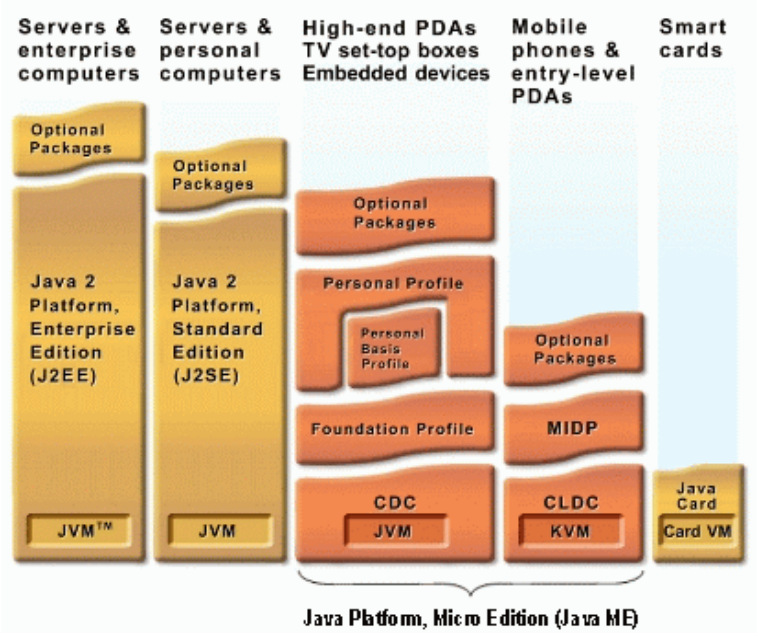


Figure 6.3: J2ME Components (Source: [SunMicrosystems, 2008a]).

When comparing to other Java development, developing for this platform requires special attention:

- Smartphones have **limited small memory**, therefore many compromises have to be done in terms of memory usage;
- **Small screen sizes** which requires careful planning of all GUI elements for greater usability;
- Alternative **input/output** methods;
- **Limited processing capacity**;
- Intense use of **profiling tools**;
- Careful **application architecture planning**;
- Use of **device emulators** and **cross-development**.

6.3.2 Bluetooth Limitations

One of the most important features of the smartphone is its communications capabilities. Among the available, the chosen communication mean for this work was Bluetooth [Bluetooth, 2008]. It consists on a protocol for radio frequency data exchange. Bluetooth communication is a great mean for wirelessly exchanging information with considerably high speed rates. Although its great capabilities, the following are some of the bluetooth's limitations:

- Interferences from other devices;
- Limited number of Bluetooth connections on smartphones;
- Energy consumption.

6.3.3 Smartphone Models

For system testing, we have used the smartphones: Nokia N80 [Nokia, 2008a] and Nokia N95 [Nokia, 2008b]. Both devices have good processing capabilities and considerable memory, when comparing with other smartphones available in the market, and can be programmed using the J2ME platform previously presented.

(a) Nokia N80 (b) Nokia N95

Figure 6.4: Smartphone models Nokia N80 and Nokia N95.

To show the differences between the two smartphone devices, we present in Table B.2 the main characteristics with interest for this project. This comparison will later help to understand the experimental results observed.

6.4 NXT Middleware

The main focus of this work consists in developing applications for an embedded system, consisting of one or more smartphones and a Lego NXT. In this system, the smartphone is responsible to execute the bulk of navigation algorithms and the Lego NXT is the physical mobile robot platform being controlled by the former. In order to provide a seamless integration between the devices, a middleware component providing a bluetooth communication layer abstraction was developed[1].

[1]Due to a similar scope of work, the information presented in this section was written and developed in conjunction with a master degree colleague, Luís Tarrataca. For more information on Luís Tarrataca's master thesis check [Tarrataca, 2008].

The main objective of the middleware component is to facilitate application development for the system composed by the smartphone and the NXT-based mobile robot. The core functionality of the middleware consists in providing simple abstractions for Bluetooth communication and also access to the mobile robot's sensors and actuators. The middleware component was developed in the Java programming language and was built on top of the leJOS firmware.

6.4.1 Middleware Component Description

By definition, middleware is a specific piece of software that interconnects and integrates software components and allows them to exchange data [Bray, 2008]. Our middleware component interconnects a server software application running on the leJOS's mobile robot, with a client software application running on a smartphone.

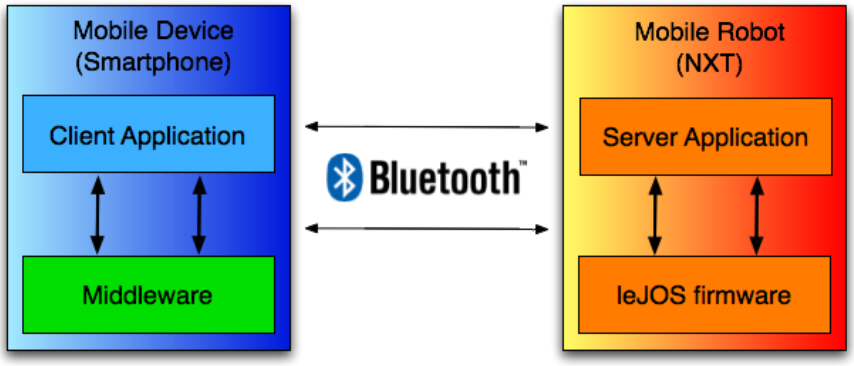


Figure 6.5: NXT Middleware Platform.

By sitting itself between the mobile client and the mobile robot (see Figure 6.5), the middleware takes responsability in all integration efforts that need to be taken care of when developing applications for this system, providing through its core API the following funcionalities:

- Abstraction to Bluetooth communication between the smartphone and the NXT Brick - Bluetooth discovery, connections and the bidirectional channel for data exchange are created in a transparent way;
- Reduce application development time - Since Bluetooth communication issues are abstracted, the application programmer only needs to worry about the specific details of its aplication's logic. The development time is reduced as well as the application's complexity;
- Access to NXT sensors and actuators - Functional requirements of the mobile robot such as the ability to do rotation, forward and backward movements, as well as sensor readings are provided through the standard API.

We now exemplify the use of the middleware through a demo application. Table 6.1 presents the main methods made available from the middleware and Figure 6.6 presents the execution flow for a client application running on a smartphone. The code starts by creating a MobileClientNXT object which, given a NXT's bluetooth address to connect to, will provide the communication abstractions with the NXT Brick as well as the NXT control commands.

Method	Description
connect()	Method that performs the connection to the NXT mobile robot.
disconnect()	Method that ceases the current connection with the NXT mobile robot.
forward(velocity)	Method that transmits to the NXT the order to move forward at a determined velocity of movement in degrees/second.
backward(velocity)	Method that transmits to the NXT the order to move backward at a determined velocity of movement in degrees/second.
rotate(angle)	Method that transmits to the NXT the order to rotate to a specific angle.
stop()	Method that transmits to the NXT the order to stop any movement that it is currently performing.
travel(distance)	Method that transmits to the NXT the order to move for a specified distance.
getSensors()	Method that requests to the NXT the list of available sensors.
getSensorValue(sensor)	Method that request a sensor measurement to the NXT.

Table 6.1: Middleware methods list.

Then, a connect command is issued in order to establish the connection between the smartphone and the NXT (Line 2 of Figure 6.6). Once the connection has been established, a forward command with a given velocity is issued, followed by a rotation command by a given angle. The next command tells the NXT to stop all actions and the execution flow is terminated with a disconnection command in order to notify the NXT that the smartphone no longer intends to maintain communication.

Name: Example Code Snippet

```
1 MobileClientNXT nxtClient = new MobileClientNXT(BluetoothAddress);
2 nxtClient.connect();
3 nxtClient.forward(velocity);
4 nxtClient.rotate(angle);
5 nxtClient.stop();
6 nxtClient.disconnect();
```

Figure 6.6: Example middleware code snippet.

6.4.2 Related Work

Besides leJOX NXJ, the leJOS development team has a Java package project designated by iCommand [Solórzano et al., 2008b] to control the NXT Brick over a Bluetooth connection. The iCommand project was released in 2006 with the objective of allowing the remote control of the NXT Brick in the Java programming language while the leJOS firmware was still in development.

They use the standard Lego NXT firmware to receive commands from Java code on a computer whilst our middleware uses the leJOS NXT firmware.

The leJOS development team has already expressed the intention to develop a new iCommand package compatible with the leJOS NXJ firmware [Solórzano et al., 2008a].

6.5 Prototype

The prototype consists on a mobile robot able to navigate through the environment with an attached smartphone. The attached smartphone is placed with its built-in camera facing the front of the robot, since it is responsible for acting as an intelligent image sensor. Besides the smartphone used as a visual sensor, another smartphone, carried by the user, can be used to control the mobile robot. The mobile robot's mobility is provided by the continuous caterpillar tracks, whose traction is powered by two independent rotation motors. The prototype size can be seen in Figure 6.7 and some photographs in Figure 6.8.

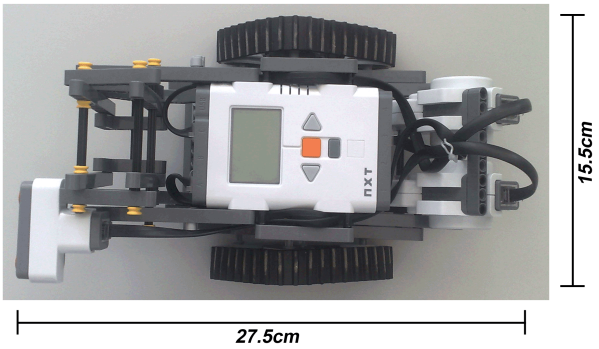
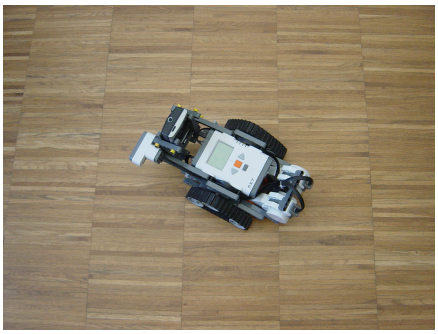
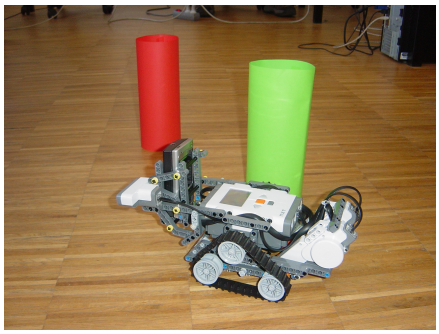


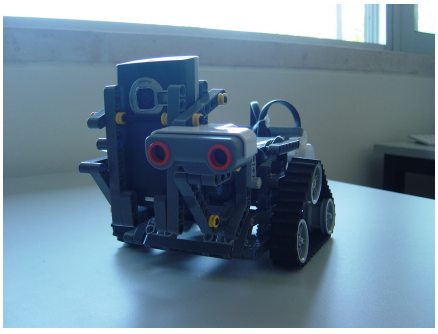
Figure 6.7: Prototype size.



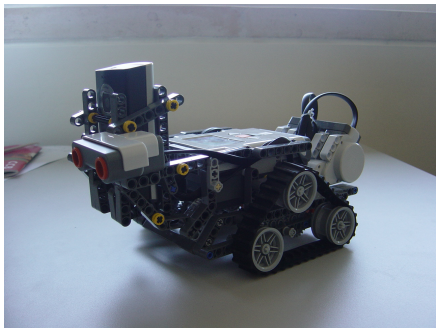
(a) Top view



(b) Side view with landmarks



(c) Front view



(d) Side view

Figure 6.8: The prototype system.

6.5.1 Application Development Cycle

The application development cycle for this system consisted on a normal iterative and incremental staged cycle common for the design, implementation and testing of embedded system software. In this thesis, this approach was used for developing Java J2ME applications for a concrete embedded system - the smartphone. The main stages are presented in Figure 6.9. One important aspect to notice is that at all stages, if any error or problem is detected, a return to a previous stage is usually performed in order to implement necessary corrections and optimizations.

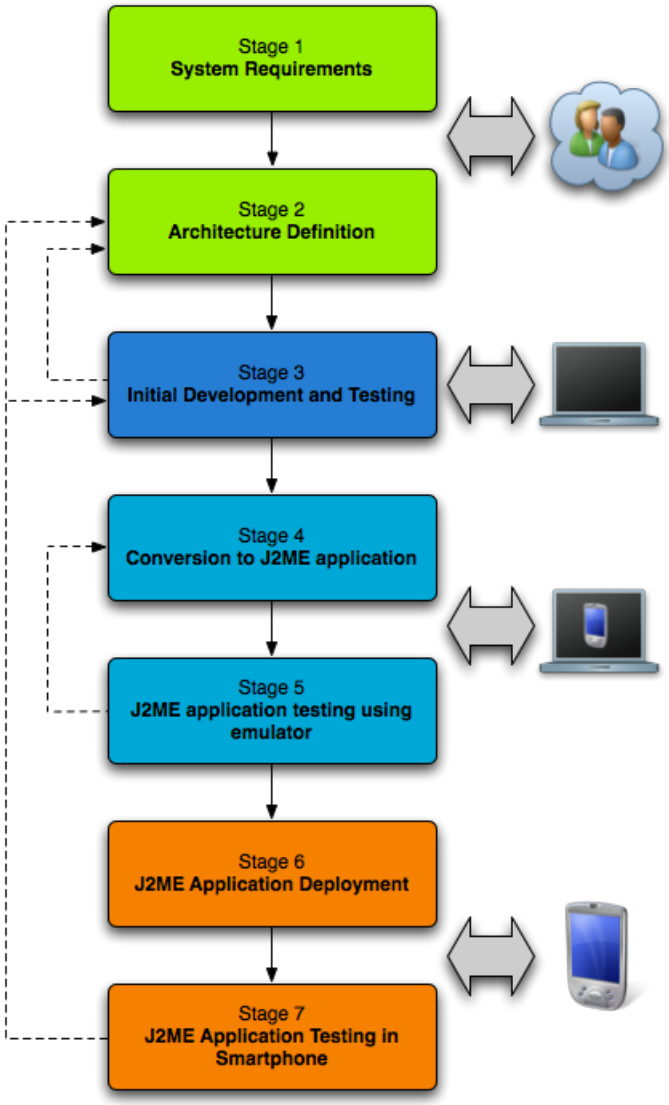


Figure 6.9: Embedded Software Development Cycle.

- Stage 1 - **System Requirements**
  The first stage is responsible for gathering and understanding the requirements intended for the system. This will establish the goals that need to be reached in order for the system to have the desired behavior. Functional and non-functional requirements need to be specified in this stage.

- Stage 2 - **Architecture Design**
  In this stage we plan the architecture to be created bearing in mind the requirements gathered and the functionalities that will fulfill them. The architecture needs to consider both the development/testing platform and the destination embedded platform. In this thesis, the software architecture designed gave relevance to modularization in order to improve independence between functional components.

- Stage 3 - **Initial Development and Testing**
  Initial development marks the translation of the designed architecture into implementation. In this stage, although always thinking of embedded systems as the destination platform, the first implementation is done using a standard computer platform. Within this development platform, the architecture is validated and the main functions and algorithms are implemented and tested.

  In order to help development and testing of specific algorithms, two simulators were also developed, the Potential Fields simulator (see Figure 6.10(b)) and the Visual Landmark Recognition simulator (see Figure 6.10(c)).

  The act of simulation generally involves representing certain key characteristics or behaviors of a selected physical system. Simulation characteristics allow changing variables, predictions and assumptions in order to preview their effect in the behavior of the system. Doing such simulations on a standard computer platform has numerous advantageous:

  - Verifying the behavior of a system before the tests on the field with the target embedded system;
  - Faster development cycles, since the initial requirements and architecture can be conceptually proven and if needed adjusted and corrected early;
  - Lower testing costs, because of less unnecessary early embedded system testing.

- Stage 4 - **Conversion to J2ME Application**
  At this point, all software implementation was done using a standard computer platform. In this stage a stable implementation has to be converted to J2ME, which involves some adjustments, since the target platform has more constraints.

  In order to simplify the process of migration from standard J2SE programming code to mobile MIDlet J2ME code, some practises were applied to ease this code translation. All the practises have in mind reducing memory footprint and increasing mobile device performance.

  - Reduce object creation and increase object reutilization;
  - Free objects by assigning them the value *null* when they are no longer necessary. This will allow the garbage collector to be more efficient and minimize its usage;
  - Restrict the use of data structures only to those supported by the J2ME API. Also, whenever possible use arrays instead of collection objects (e.g., Vector class) since operations on them are much faster and they are more memory efficient;
  - Remove unused methods, classes and libraries. Only the resources actually used should be included;
  - Reduce the application size by using an obfuscator.

- Stage 5 - **Smartphone Emulation**
  Using emulation of smartphone characteristics is extremely important because prior to deployment, smartphone behavior can be simulated and therefore many experimental results and performance conclusions can be taken. For performance evaluation, the process of profiling enabled the extraction of important execution information which was very important for code optimization.

  In concrete, smartphone emulation was done using the general purpose emulators provided in Sun's Java Wireless Toolkit [SunMicrosystems, 2008b] (see Figure 6.10(a)). These emulators provide the common characteristics that are present in the great majority of the current existing smartphones.

- Stage 6 - **J2ME Application Deployment**
  After all software developments become stable while executing in emulated and simulated platforms, the applications need to be deployed to the real embedded system device so that further and more correct testing and execution can be performed.

- Stage 7 - **J2ME Application Testing in Smartphones**
  This stage is very important because now the software developed is in its final destination platform and therefore needs to have its behavior evaluated. While testing, functionality needs to be validated against the initial defined requirements, to verify if all goals intended were reached.

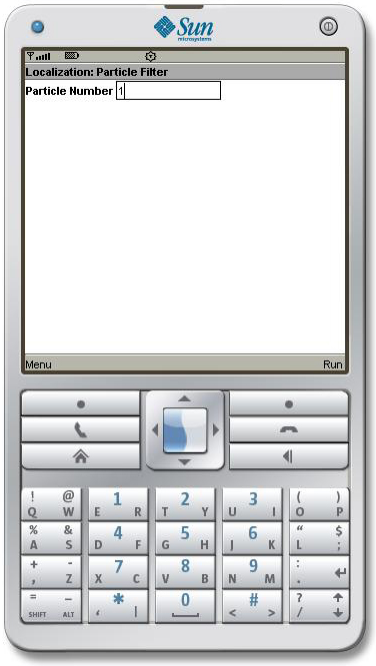
6.5.2 Experiments on the field

The development of the software applications takes care mostly of the smartphone component of the system and the software part of the mobile robot component. First, the navigation algorithms are implemented, tested and their performance evaluated, but on a simulation basis. Experiments in real environments are needed to validate and visualize the system's behavior and the capacity to act in day-to-day life situations with real interactions and physical beings. Chapter 7 will address directly this issue.

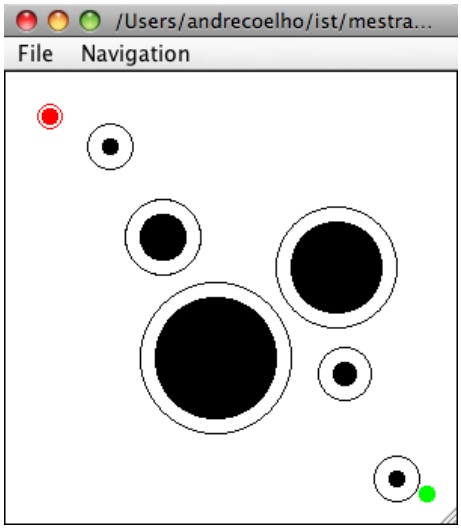
6.5.3 Distributed Smartphones

One solution tested in this system was the use of more than one smartphone for more computationally demanding tasks. Because of some of the algorithm's processing and memory usage, functional distribution was a reliable way of achieving the desired results while allowing complexity distribution and functional separation.

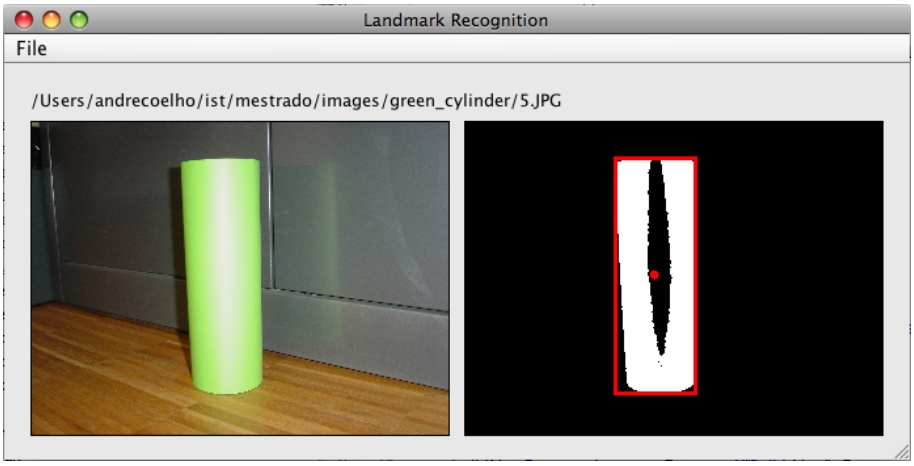
In this project, we preferred to have one smartphone, acting solely as an intelligent visual sensor and a second smartphone that would act as the navigation algorithmic processing unit. The visual sensor smartphone can therefore concentrate on providing observations with added preprocessing functionalities for image treatment and setup, and sending to the processing smartphone the results of identification of landmarks (beacons), for instance. These tasks no longer need to exist in the primary processing smartphone, relieving it to other tasks or speeding up the overall execution.



(a) Sun's WTK Emulador



(b) Potential Fields Simulator



(c) Visual Landmark Recognition Simulator

Figure 6.10: Sun's WTK emulator (a) and Simulators used for development (b-c).

6.6 Summary

This chapter presents the implementation characteristics for the embedded system prototype. The main components of the system, the mobile robot and the smartphone, are presented and their most important aspects explained. Next, the complete system is rationalized giving attention to the development approach and other considerations regarding a final prototype for demonstration.

Chapter 7

Experimental Results

"In theory, there is no difference between theory and practice.
In practice, there is."
Jan L. A. van de Snepscheut

7.1 Introduction

In this chapter, we present and discuss experimental results for all the navigation problems considered (e.g., Mapping, Localization and Path Planning). Here we evaluate the performance of the algorithms developed, an execution comparison between the used smartphones and the standard desktop computer, as well as the feasibility for real-time robot control.

- **Performance monitoring with profiling results**
  Profiling was done on the desktop PC running the applications with the MIDP emulator. We used the Java WTK method profiler which highlights application bottlenecks by showing the methods an application called, how often they were called, and the execution time spent in each of those methods. The WTK also provides a memory monitor which shows the memory a MIDlet uses as it runs and the current memory in use for each Java object created.

  For comparison results, of time and memory, the experiments on the PC ran on an AMD Athlon 64 X2 Dual Core Processor at 2.20 GHz with 960MB of RAM and running Windows XP SP3.

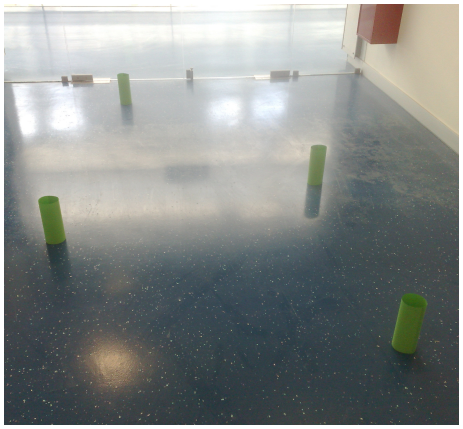
- **Field Tests**
  Field tests are a subset of experiments conducted that evaluate the performance and behavior of the implemented navigation algorithms when bundled with the mobile robot. A control layer is added to issue commands to the mobile robot.

  Our experiments consist on:

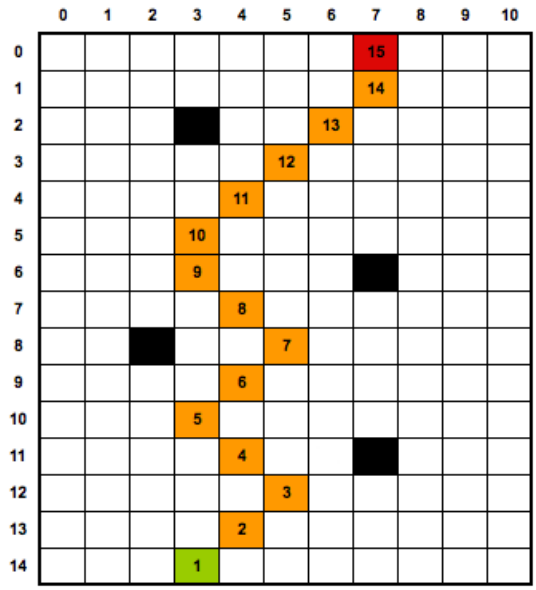
  1. Map an area of the environment while the mobile robot navigates on a predefined path.
  2. Localize the mobile robot within the environment whilst navigating on a predefined path.
  3. Navigate from a start to a goal point, avoiding all the existing obstacles in the environment.

The predefined path used for both Mapping and Localization tests allows the robot to move through free spaces in the environment, having a free view of the artificial landmarks that were previously placed, and also providing us with the possibility for correct verification of the robot's and landmarks's location.

Figure 7.1(a) shows a picture taken from the prepared environment with a total size of 330 $cm \times 450$ cm. Figure 7.1(b) shows a grid map representation of size 11×15 from the same environment, where we can see the real position of the landmarks, which are posing as obstacles. The path designed for the mobile robot is represented in orange. The grid map has a 1:30 size ratio in comparison with the real environment. This ratio allows for a grid cell size that can fully contain the mobile robot.



(a) Real Environment



(b) Grid Map environment representation with landmarks posing as obstacles, robot position, goal position and predefined path

Figure 7.1: Field environment for testing.

7.2 Mapping - Visual Landmark Recognition

Mapping experiments will try to map the environment presented in Figure 7.1, estimating the location of all landmarks posing as obstacles. The Visual Landmark Recognition approach is used, having the smartphone's built-in camera as the vision sensor. The smartphone also controls the mobile robot's movement.

First let's start by profiling the execution of the approach implemented. Figure 7.2 exhibits time execution percentages for one landmark recognition. Table 7.1 represents the absolute execution time and memory used, for comparison between the desktop PC and the smartphone models Nokia N80 and Nokia N95.

In this test, all the steps needed to identify a landmark are executed. First an image is captured, color segmentation is performed to detect the presence of a landmark, salt and pepper noise is reduced using a filter and by calculating the boundaries of the detected landmark, it is calculated the distance and orientation

from the camera's position to the position of the landmark.

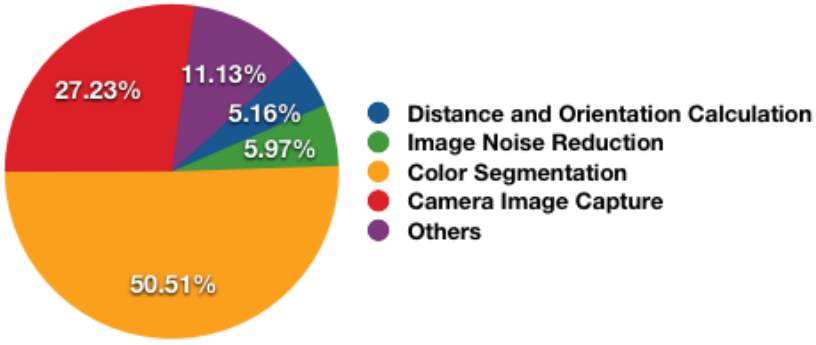


Figure 7.2: Contribution to the overall execution time of each step associated to the Visual Landmark Recognition.

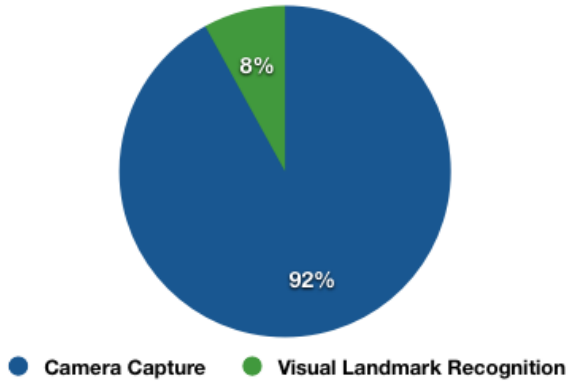


Figure 7.3: Time comparison between image acquisition and the Visual Landmark Recognition algorithm.

By observing the percentages presented in Figure 7.2 we can see that color segmentation stage dominates time consumption, being followed by the camera's image capture. Color segmentation is a meticulous process, since it needs to analyze each pixel of the image in order to evaluate if it belongs to a landmark or not. Capturing an image is not very fast and much time is spent with this stage. Let's not forget that this image capture is done by the WTK emulator, which tries to emulate a general purpose mobile phone. The smartphone models used are even slower to acquire an image from the built-in camera as can be seen from Figure 7.3. The application of the noise reduction filter and the distance and orientation calculations both consume similar times, around 5%. The Others class represents auxiliary stages for data preparation.

	PC	Nokia N80	Nokia N95
Time (ms)	453.00	3079.40	5824.30
Memory ($bytes$)	360368.00	163557.60	157840.80

Table 7.1: Time and Memory measurements for the Visual Landmark Recognition method.

Observing Table 7.1, obviously, the PC is the fastest to execute the application. Comparing the two smartphones, memory use is similar but the execution time is slower on the Nokia N95 model. The N95 has a more complex built-in camera with higher resolution, making it slower when capturing an image. Nokia N80 on the other hand has a more simple built-in camera and can capture images faster.

Now let's consider a field experiment with the execution of the implementation on the environment from Figure 7.1. For each step the mobile robot does it captures an image and tries to identify if in that image, there are landmarks present. The process will be repeated until the mobile robot reaches its final position. Nokia's N95 model is used in this experiment.

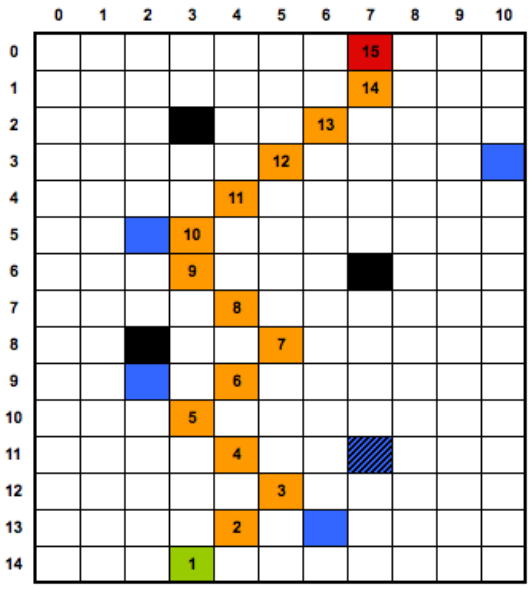


Figure 7.4: Grid map environment representation with the landmarks considering real and estimated positions.

Figure 7.4 presents, in blue, the grid cells which are the estimates of the landmark positions from the mobile robot's observations. We can notice that the mobile robot identified five landmarks, with only one in the correct cell position. All others presented errors in the distance and orientation estimation.

Two important aspects justify these results:

1. Errors resulting from distance and orientation calculation, which are mainly caused by an incorrect color segmentation stage due to differences in illumination, shadows and other environment aspects that change the perception of color;

2. Odometry errors from the mobile robot's movement.

It is important to recall that the implemented landmark recognition does not consider captured images with more than one landmark.

We can conclude that although the implemented recognition makes use of a simple approach to computer vision, and in previous tests considering good lighting conditions, the identification of the landmarks were considerably accurate. In field testing, the method revealed less accurate. The robot's movement and variable lighting conditions prevent the method from achieving its best results. This solution, through experiments cannot be considered a very reliable method for accurate mapping purposes in real-time mobile robot navigation.

7.3 Localization - Particle Filter

Experimental results for Localization were conducted considering only the global localization approach introduced in Chapter 4. For the field experiments, the particle filter uses as a measurement model, the visual landmark recognition presented in Chapter 3.

First off let's start by profiling the Particle Filter implementation. For this test we considered one execution of the method with a total number of 1000 particles and using the environment presented in Figure 4.5. The robot pose estimate is only performed at the end of the mobile robot's movement.

Figure 7.5 presents the percentages of execution time of the main phases of the Particle Filter method. Table 7.2 presents the execution time and memory used comparison between running the implemented localization application on a standard desktop PC and on the smartphone models Nokia N80 and Nokia N95.

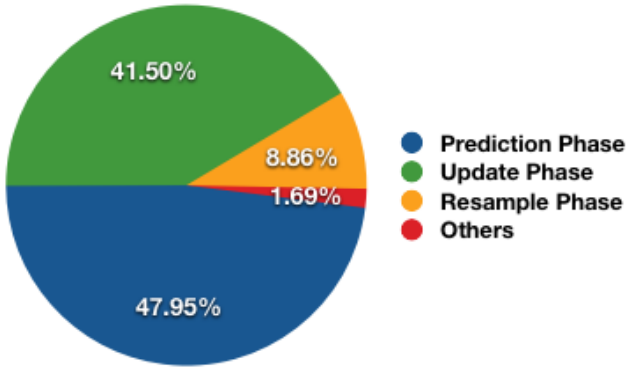


Figure 7.5: Contribution to the overall execution time of each phase of the Particle Filter method.

	PC	Nokia N80	Nokia N95
Time (ms)	78.00	3618.00	1725.00
Memory ($bytes$)	139200.00	138060.00	78212.00

Table 7.2: Time and Memory measurements for the Particle Filter method.

According to the experiments, the phase which was responsible for the highest percentage of execution time was the Prediction Phase with almost 48%. The Update phase followed with more than 41%. Finally and considering the number of particles used and their distribution within the environment, the Resample Phase took approximately 9% of execution time.

In the Prediction Phase the particle's movement and noise addition are the main consuming areas. Within the Update Phase, the execution time is dominated by the measurement of the particles. For the Resample Phase, the execution time is dominated with the selection of particles to remove and particles to duplicate. The last 1.69% of execution time is spent by auxiliary tasks as well as attainment of the mobile robot's pose estimate.

Now let's consider testing experiments on the field that will try to localize the mobile robot in the environment presented in Figure 7.1. The Particle Filter approach will be executing on the Nokia's N95 smartphone, and it will consider the use of 1000 particles. This smartphone will also control the mobile robot's motion. The measurement model will be a visual sensor performed with the Visual Landmark Recognition method running

on the Nokia N80 model.

Results for the execution of this field experiment are presented in Table 7.3. Consider that the positions are given as xy position and θ orientation: $[x; y; \theta]$. The robot's real position at the end of the predefined path is $[7; 0; 90°]$. The pose estimate error (distance to the correct position) is given by Equation 4.7.

	Experiment Number				
	#1	#2	#3	#4	#5
Best Particle Position	$[4; 0; 90°]$	$[7; 0; 90°]$	$[9; 1; 90°]$	$[4; 0; 90°]$	$[0; 11; 180°]$
Pose Estimate Error	3	0	2.24	3	13.04

Table 7.3: Localization experimental results with different particle numbers.

By analyzing Table 7.3, we can observe that only one of the experiments estimated the robot to be at its exact physical location. In the other four experiments, three were relatively close to the robot's real position, and the last one was very far from the robot's position.

We can easily conclude that the problems identified in Chapter 4, here in field experiments are very visible. The random initialization of the particles makes the method difficult to predict, which makes it provide very different results on different runs of the algorithm. For the same map, same measurement model and motion model, the algorithm can provide a very good estimate of the mobile robot position or a very bad one. One possible solution to this problem is the increase of the number of particles, but with high additional computational costs.

The solution implemented was feasible to be executed in real-environments, but the mobile robot cannot be moving at high speeds. The visual sensor, the particle filter execution and the transmission of information from the two smartphones using Bluetooth are time demanding and cannot, without further optimizations, be used to navigate mobile robots at high speed. Nevertheless, considering a slower motion, this solution was able to provide a mechanism for mobile robot localization.

7.4 Path Planning - Potential Fields

The performance evaluation for the Potential Fields algorithm for Path Planning was done considering the environment shown on Figure 5.17. In this implementation of the Potential Fields method, there is a previous stage of loading the environment map to feed the algorithm. In our implementation, the environment map is provided through a XML input file (for the XML format specification see Appendix A). The detection of the local minima locations is done considering the lookahead approach and the escape technique used is the Virtual Obstacle Escape technique.

In Figure 7.6 we present a comparison of the map's loading and processing time between a PC and the two Nokia smartphone models. As can be seen from Figure 7.6, the PC is the fastest of the three to load and process the XML file. The Nokia N95 model is faster than the N80 model since it is equipped with a faster processor.

With the environment map loaded and processed, the potential fields algorithm can initiate. We can then define the normal execution as divided into two stages: the direction of robot movement through the potential calculations and the detection of local minima. As was explained in Chapter 5, the detection of local minima

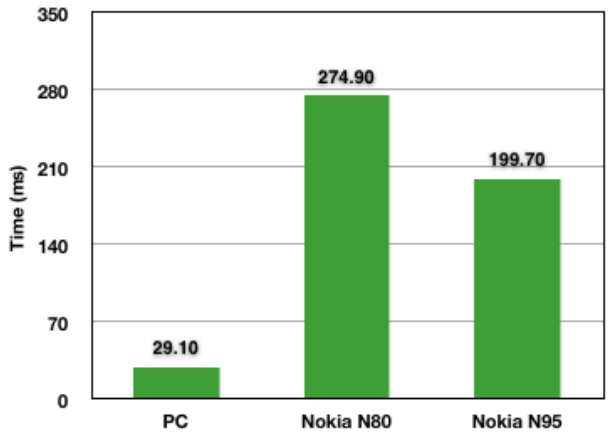


Figure 7.6: Average time to load the XML environment map.

is done rather often in order to provide a smoother robot movement, detecting these locations before being trapped in them. Figure 7.7 shows the percentage of the overall execution time for these two stages. Table 7.4 presents the time and memory comparison of the execution of the algorithm on the PC and on the Nokia smartphone models.

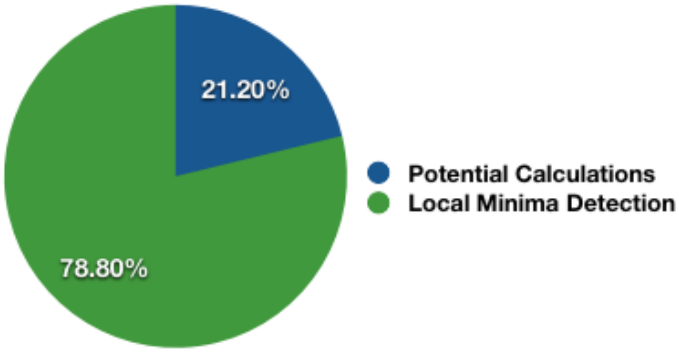


Figure 7.7: Contribution to the overall execution time of each stage of the Potential Fields.

As can be seen in Figure 7.7, the detection of local minima locations, performed with the lookahead approach, is time consuming and depending on the lookahead steps can compromise the execution of the whole algorithm. The information presented in Figure 7.7 assumes a lookahead of five positions. Although the detection of the local minima also involves potential calculations, the 21.20% shown for Potential Calculations in the profile chart are only relative to those used for mobile robot movement.

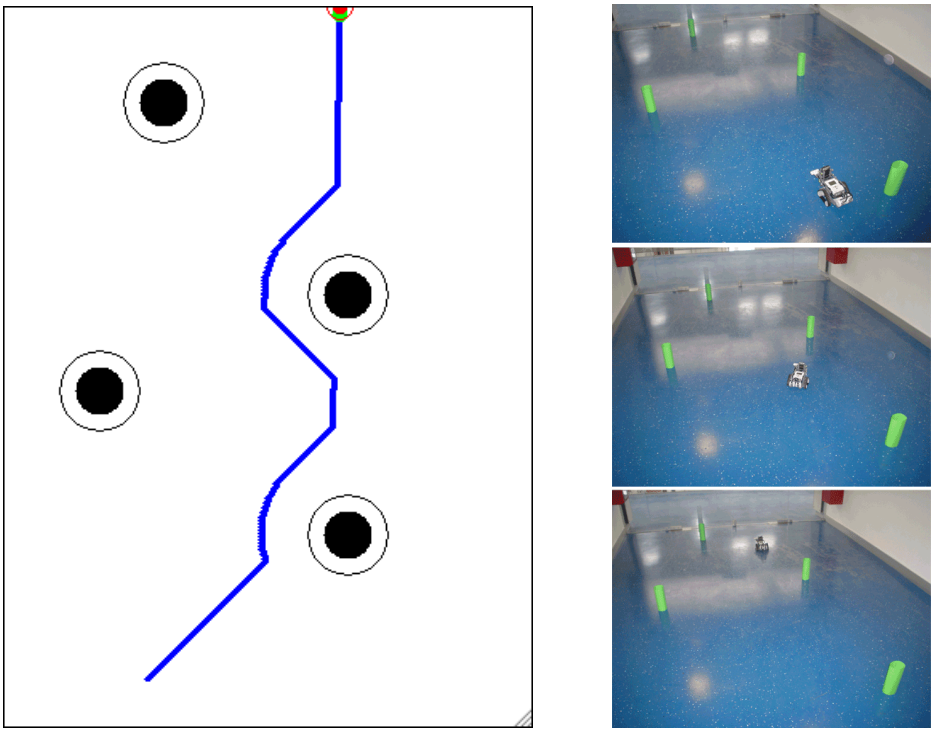
	PC	Nokia N80	Nokia N95
Average Step Time (ms)	11.00	377.00	278.75
Total Time (ms)	7664.60	253442.75	187882.75
Total Memory ($bytes$)	705572.00	108412.00	109437.00

Table 7.4: Time and Memory measurements for the Potential Fields algorithm.

Similar to the results from the other algorithms performed before, here also the PC presents the lowest execution time. When considering algorithms that simply need to accomplish fast calculations, the Nokia N95 has faster execution times than the Nokia N80 model.

Considering now the Path Planning field experiment, we conducted the test in the environment presented in Figure 7.1. The Potential Fields approach will provide a clear path from the start position to the goal position. The algorithm and the mobile robot control were both executed by the Nokia N80 smartphone.

Figure 7.8 shows both the simulation of the path that the mobile robot took, and some photographs taken in the environment while the mobile robot was navigating. Figure 7.8(a) shows also the internal perception of the environment and the path taken by the mobile robot in this test. In this environment set, the potential fields algorithm for path planning provided a clear path, free from any collision with obstacles.



(a) Potential Fields simulation

(b) Sequence of navigation images

Figure 7.8: Experiment in the field for Path Planning with Potential Fields.

On the field, the path was similar to the path presented for simulation having to consider movement errors from odometry. Considering the mobile robot moving at the maximum possible speed, the field execution had the following characteristics:

- XML map loading time of 181 ms;
- Execution time, considering the movement through the environment, of around 278 s.

In the field test, due to the nature of the Potential Fields method, the robot revealed some strange orientation changes when avoiding obstacles. This fact was never very noticeable in simulation testing. We concluded that, even in the absence of local minima locations, some raw directional vectors cannot be directly applied for the robot's movement. Some of these directional vectors force the robot to perform expensive rotations that need to be smoothen beforehand. For field testing, a preprocessing translation model should be applied in order to better convert the output from the potential fields to the mobile robot's movements.

7.5 Summary

In this chapter the experimental results obtained by the three navigation problems are presented. The experimental results shown include profiling results and experiments on the field. With the profiling results we were able to identify the overall contribution from each stage in the tested algorithms. With this discrimination of execution time we are more aware of the internal behavior of the algorithms.

Field experiments are the main contribution of this chapter. Simulations allow us to easy test and deploy our applications, but in terms of mobile robot navigation, we can only really conclude how good the solution is in field testing. The implemented algorithms for navigation (e.g., Mapping, Localization and Path Planning), when executed on the field, revealed less accurate than on simulations, providing worse overall results. It is clear that for better results in field testing, more experiments have to be conducted, earlier in the development.

Chapter 8

Conclusions

"A conclusion is the place where you got tired of thinking."
Arthur Bloch

Mobile robot navigation is today a major area in the robotics field. It approaches three of the most important mobile robot problems: Mapping, Localization and Path Planning. This thesis researches the use and feasibility of a system, composed by a mobile robot (Lego's Mindstorms NXT) and a smartphone (Nokia N80 or Nokia N95) which communicate using bluetooth technology. Together, they represent a robotic set that is remotely controlled by the smartphone, which guarantees the execution of the navigation algorithms.

The smartphones used were both able to execute all the applications developed, being the Nokia N95 the fastest of the two. All programming was developed using Java's J2ME for embedded systems. J2ME, although more limited than the standard Java version, still provides the basis that is needed for the implementation of the selected algorithms. Constraints found on the implementation were surpassed using common embedded system programming rules, which allowed to keep the consistency of the algorithms. Important to note is the use of the Wireless ToolKit (WTK) emulator, that was the application's testing model. It provided fast code evaluation, thus reducing the number of deployments made to the smartphone models.

The mobile robot kit used is a very multifaceted equipment, as it allows to build a vast type of robots. For precise navigation, the construction nature of the kit made it sometimes difficult to use, due to inherent odometry errors. Experiments on the field evidenced those errors.

Communication between the system components using bluetooth was a very effective method, which alongside the developed middleware was an easy and reliable way to transfer information between components.

Mapping using artificial landmarks revealed easy to detect with the smartphones built-in camera. Unfortunately, there are two stages that limit the overall quality of the detection:

- First, the image capture time for a smartphone is time consuming which slowed the execution of the method.

- Second, the calculation of distance and orientation from the robot to the landmark are very dependent of the quality of the color segmentation stage output. Therefore, in conditions where illumination changes often, a less correct color segmentation output can cause incorrect measurements of distance and orientation.

Although these two constraints, the solution presented succeeded on identifying landmarks with acceptable memory usage and some expensive execution time.

For Localization we used the Particle Filters approach. The results were worst than expected due to the limitations encountered in the implemented measurement model. The measurement model revealed unable to provide sufficient information to distinguish similar locations. With a more correct and precise measurement model, surely the results would have been greatly improved. Nevertheless, experimental results were conducted within different environments, in simulation and on the field, and the outcome was always in positions very similar to the actual mobile robot location.

Considering our implementation of Path Planning as Potential Fields, based on evaluations and tests performed, we conclude that although the approach may not always ensure the shortest path for robot motion, it still provides an obstacle-free path with low execution times and memory usage. The lookahead function implemented allowed smoother paths for motion and, together with escape techniques, achieved a solution for detecting and avoiding the local minima problem. Unfortunately the lookahead function is also the main performance bottleneck in this method. The Virtual Obstacle Escape was the most successful escape technique from local minima locations.

The work addressed by this thesis can broaden the research on robotics control using mobile phone devices which are today one embedded system that most of us cannot live without. The smartphone's feasibility for the execution of navigation algorithms, testifies the current processing capacity's state in high-end mobile phone devices. Hopefully, this research will induce others to continue to explore this possibility and create more advanced and complex services, applications and joint robotic systems that can help people in their daily tasks.

8.1 Future Work

Future work should explore other possibilities and routes in which this thesis's work could head to. Main future work additions could be:

- **Further optimization of the implemented algorithms**
  Using code profiling and considering programming rules for embedded systems, the current implementation of the navigation algorithms could be optimized in depth with the goal of providing better performance which is needed when environment complexity increases.
- **Implementation of other navigation algorithms**
  The system used has implemented one solution for each of the main navigation problems. These solutions were chosen considering their complexity, outcome and possible adaptation for an embedded version. Other solutions were considered but they were not implemented. Some of these other solutions (e.g., SLAM, Extended Kalman Filters, Ant Colony) should be implemented for comparison and to verify their outcome on experimental testing.

- **Further experimental testing on the field considering more complex environments**
  Experiments on the field were conducted but within limited sized and simple environments, always considering static obstacles. Future testing should include larger and more complex environments, possibly with inclusion of moving obstacles. Also, field testing should be incorporated earlier in the development process. Although field testing is time consuming, the overall result can be greatly improved.
- **Identification of more than one landmark per captured image**
  The landmark recognition using computer vision was implemented with the restriction on the number of landmarks that can be identified in a captured image. Future work should consider the identification of multiple landmarks within a captured image.
- **Particle Filter measurement sensors**
  As seen for the Particle Filter implementation, the implemented version had its results limited by the measurement sensor used. Other measurement sensors need to be considered to see which one can provide better information to help distinguish similar locations (e.g., using a laser range sensor mounted on the mobile robot could greatly improve the field results obtained).
- **Potential Fields local minima lookahead study**
  A cost/benefit study should be performed to evaluate the impact of the number os lookaheads in the Potential Field approach, when trying to detect local minima locations. This study should identify rules to establish the more correct value of lookahead needed, for smooth movement with avoidance of such problematic locations.

8.2 Applications

The work presented in this thesis has many possible real life applications. The possibilities are only limited by our imagination and creativity. We list next some applications for this work:

- **Create a new generation of cheaper, more intelligent mobile robots**
  Nowadays, small robotic systems are a part of our day to day lives. They have appeared as autonomous vacuum cleaners, pool cleaners, floor washing, etc. Considering the use of smartphones as a processing unit, these robots can see their processing capabilities reduced making their production cheaper and less complex, which therefore reduces also their price, making them more accessible to the end users.

  We can surely imagine a simple vacuum cleaner that uses our mobile phone as its processing unit which provides it with the map of the house and personalized instructions for when, how and where to vaccum.
- **Enhance the role of smartphones as a control unit**
  Embedded systems are extremely present in our houses, vehicles and wherever we go. They exist on traffic lights, sport scoreboards and movie theaters, just to name a few. Smartphones, which are an indispensible device nowadays, could assume the role of controlling most of these embedded systems, providing us with personalized content and behavior.

  Imagine arriving at home after a day's work, and automatically and autonomously having the mobile phone in our pocket detect the presence of other embedded systems and by knowing our personal preferences and mood, could set the tv channel and volume, change the lights strength and activate the garden watering system.

Bibliography

[Baczyk et al., 2003] Baczyk, R., Kasinski, A., and Skrzypczynski, P. (2003). Vision-based mobile robot localization with simple artificial landmarks. *Prepr. 7th IFAC Symp. on Robot Control, Wroclaw*, pages 217–222.

[Bagnall, 2007] Bagnall, B. (2007). *Maximum Lego NXT: Building Robots with Java Brains*. Variant Press.

[Bluetooth, 2008] Bluetooth (last visited in July 2008). Bluetooth.com - the official bluetooth technology info site. `http://www.bluetooth.com/`.

[Bray, 2008] Bray, M. (last visited in June 2008). Middleware: Software technology roadmap. `http://www.sei.cmu.edu/str/descriptions/middleware_body.html`.

[Bruce et al., 2000] Bruce, J., Balch, T., and Veloso, M. (2000). Fast and inexpensive color image segmentation for interactive robots. *In Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '00)*, 3:2061–2066.

[Carnegie Mellon, 2008] Carnegie Mellon, U. (last visited in February 2008). Tartan racing @ carnegie mellon university. `http://www.tartanracing.org/`.

[Chikamasa, 2008] Chikamasa, T. (last visited in July 2008). nxtosek ansi c/c++ with osek rtos for lego mindstorms nxt. `http://lejos-osek.sourceforge.net/`.

[Cormen and Rivest, 2001] Cormen, T. H. and Rivest, R. L. (2001). *Introduction to Algorithms*. MIT Press, 2nd edition.

[Dellaert et al., 1999] Dellaert, F., Fox, D., Burgard, W., and Thrun, S. (1999). Monte carlo localization for mobile robots. *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 2:1322–1328.

[Dorigo, 1992] Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy.

[Foster-Miller Inc., 2008] Foster-Miller Inc., Q. N. A. (last visited in February 2008). Talon military robots, eod, swords, and hazmat robots. `http://www.fostermiller.com/lemming.htm`.

[Fox, 1998] Fox, D. (1998). *Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation*. PhD thesis, Institute of Computer Science III, University of Bonn, Germany.

[Fox et al., 1999] Fox, D., Burgard, W., and Thrun, S. (1999). Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427.

[Ge and Cui, 2000] Ge, S. S. and Cui, Y. J. (2000). New potential functions for mobile robot path planning. *IEEE Transactions on Robotics and Automation*, 16(5):615–620.

[Goodrich, 2002] Goodrich, M. A. (2002). Potential fields tutorial. *Class Notes*.

[Gordon et al., 1993] Gordon, N., Salmond, D., and Smith, A. (1993). Novel approach to nonlinear/non-gaussian bayesian state estimation. *Radar and Signal Processing, IEE Proceedings F*, 140(2):107–113.

[Guibas and Motwani, 1995] Guibas, L. J. and Motwani, R. (1995). The robot localization problem. In Goldberg, Halperin, Latombe, and Wilson, editors, *Algorithmic Foundations of Robotics, The 1994 Workshop on the Algorithmic Foundations of Robotics, A. K. Peters*.

[Hansen, 2008a] Hansen, J. (last visited in July 2008a). Next byte codes & not exactly c. `http://bricxcc.sourceforge.net/nbc/`.

[Hansen, 2008b] Hansen, J. (last visited in July 2008b). Not quite c. `http://bricxcc.sourceforge.net/nqc/`.

[Hu et al., 2004] Hu, W., Downs, T., Wyeth, G., Milford, M., and Prasser, D. (2004). A modified particle filter for simultaneous robot localization and landmark tracking in an indoor environment.

[iRobot Corporation, 2008] iRobot Corporation (last visited in February 2008). `http://www.irobot.com/`.

[Jang et al., 2002] Jang, G., Lee, S., and Kweon, I. (2002). Color landmark based self-localization for indoor mobile robots. *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, 1:1037–1042.

[Kalman, 1960] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45.

[Khatib, 1986] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98.

[Kim and Khosla, 1992] Kim, J.-O. and Khosla, P. (1992). Real-time obstacle avoidance using harmonic potential functions. *IEEE Transactions on Robotics and Automation*, 8(3):338–349.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *The American Association for the Advancement of Science*, 220(4598):671–680.

[Koren and Borenstein, 1991] Koren, Y. and Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation. *Proceedings of the IEEE Conference on Robotics and Automation*, 2:1398–1404.

[Lee, 2004] Lee, L.-F. (2004). Decentralized motion planning within an artificial potential framework (apf) for cooperative payload transport by multi-robot collectives. Master's thesis, Department of Mechanical and Aerospace Engineering.

[Lego, 2006] Lego (2006). Lego mindstorms nxt - hardware developer kit. Technical report, The Lego Group.

[Lego, 2008b] Lego (last visited in February 2008b). `http://mindstorms.lego.com/`.

[Lego, 2008a] Lego (last visited in July 2008a). Mindstorms nxt software. `http://mindstorms.lego.com/overview/NXT_Software.aspx`.

[Leonard and Durrant-Whyte, 1991] Leonard, J. J. and Durrant-Whyte, H. F. (1991). Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382.

[Liu et al., 2005] Liu, G., Li, T., Peng, Y., and Hou, X. (2005). The ant algorithm for solving robot path planning problem. *Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05)*, 2:25–27.

[Lowe, 1999] Lowe, D. G. (1999). Object recognition from local scale-invariant features. *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, 2:1150–1157.

[Matarić, 2007] Matarić, M. J. (2007). *The Robotics Primer*. The MIT Press, 1st edition.

[MathWorks, 2008] MathWorks (last visited in July 2008). Programming lego mindstorms nxt robotics with matlab and simulink. `http://www.mathworks.com/programs/lego/`.

[Montemerlo et al., 2002] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). Fastslam: A factored solution to the simultaneous localization and mapping problem.

[NASA, 2008] NASA (last visited in February 2008). Mars exploration rover mission. `http://marsrovers.nasa.gov/home/index.html`.

[Negenborn, 2003] Negenborn, R. (2003). Robot localization and kalman filters. Master's thesis, Utrecht University, Netherlands.

[Nokia, 2008b] Nokia (last visited in February 2008b). Nokia nseries n95 smartphone. `http://www.nseries.com/products/n95/`.

[Nokia, 2008a] Nokia (last visited in June 2008a). Nokia nseries n80 smartphone. `http://www.nseries.com/products/n80/`.

[Park et al., 2001] Park, M. G., Jeon, J. H., and Lee, M. C. (2001). Obstacle avoidance for mobile robots using artificial potential field approach with simulated annealing. *Industrial Electronics - Proceedings - ISIE 2001 - IEEE International Symposium*, 3:1530–1535.

[Park and Lee, 2003] Park, M. G. and Lee, M. C. (2003). Artificial potential field based path planning for mobile robots using a virtual obstacle concept. *Proceedings of the 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2003)*, 2:735–740.

[Prasser and Wyeth, 2003] Prasser, D. and Wyeth, G. (2003). Probabilistic visual recognition of artificial landmarks for simultaneous localization and mapping. *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, 1:1291–296.

[Rekleitis, 2003] Rekleitis, I. (2003). *Cooperative Localization and Multi-Robot Exploration*. PhD thesis, School of Computer Science, McGill University, Montréal.

[Rekleitis et al., 2003] Rekleitis, I., Dudek, G., and Milios, E. (2003). Probabilistic cooperative localization and mapping in practice. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 2, pages 1907–1912.

[Se et al., 2001] Se, S., Lowe, D., and Little, J. (2001). Vision-based mobile robot localization and mapping using scale-invariant features. *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, 2:2051–2058.

[Smith et al., 1990] Smith, R., Self, M., and Cheeseman, P. (1990). Estimating uncertain spatial relationships in robotics. In *Autonomous robot vehicles*, pages 167 – 193. Springer-Verlag New York, Inc., New York, NY, USA.

[Solórzano et al., 2008a] Solórzano, J., Bagnall, B., Stuber, J., and Andrews, P. (last visited in June 2008a). lejos: Java for lego mindstorms. `http://lejos.sourceforge.net/`.

[Solórzano et al., 2008b] Solórzano, J., Bagnall, B., Stuber, J., and Andrews, P. (last visited in June 2008b). lejos: Java for lego mindstorms - icommand technology. `http://lejos.sourceforge.net/p_technologies/nxt/icommand/icommand.php`.

[Stanford, 2008] Stanford, U. (last visited in February 2008). Potential fields for lane keeping and vehicle control, http://ddl.stanford.edu/pf.

[SunMicrosystems, 2008b] SunMicrosystems (last visited in July 2008b). Sun java wireless toolkit for cldc. `http://java.sun.com/products/sjwtoolkit/`.

[SunMicrosystems, 2008a] SunMicrosystems (last visited in May 2008a). Java me technology. `http://java.sun.com/javame/technology/`.

[Tarrataca, 2008] Tarrataca, L. (2008). Gesture recognition system to command robots using smartphones. Master's thesis, Instituto Superior Técnico.

[Thrun et al., 2005] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. The MIT Press.

[Veelaert and Bogaerts, 1999] Veelaert, P. and Bogaerts, W. (1999). Ultrasonic potential field sensor for obstacle avoidance. *IEEE Transactions on Robotics and Automation*, 15(4):774–779.

[VEXRobotics, 2008] VEXRobotics (last visited in July 2008). Vex robotics is a mark of innovation first, inc. `http://www.vexrobotics.com/`.

[Volpe and Khosla, 1990] Volpe, R. and Khosla, P. (1990). Manipulator control with superquadratic artificial potential functions: Theory and experiments. *IEEE Transactions on Systems, Man and Cybernetics*, 20(6):1423–1436.

[Welch and Bishop, 2006] Welch, G. and Bishop, G. (2006). An introduction to the kalman filter. Technical report, Department of Computer Science, University of North Carolina at Chapel Hill, USA.

[Yoon et al., 2001] Yoon, K.-J., Jang, G.-J., Kim, S.-H., and Kweon, I. S. (2001). Fast landmark tracking and localization algorithm for the mobile robot self-localization. *in IFAC Workshop on Mobile Robot Technology*, pages 190–195.

Appendix A

Potential Fields XML Input

A.1 Format Specification

Potential field functions need as input the location of obstacles, target goal and robot. This information is introduced by providing an XML input file that contains the following information:

- Environment dimensions;

```
<Scenario width="?" height="?">
...
</Scenario>
```

- Robot and Target Goal position;

```
<Robot x="?" y="?"/>
<TargetPoint x="?" y="?"/>
```

- A list of obstacles with definition of type, size and position.

```
<ObstacleList>
<RectangularObstacle x="?" y="?" width="?" height="?"/>
...
<CircularObstacle width="?" height="?" x="?" y="?"/>
</ObstacleList>
```

With this set of information we have data to internally represent the environment in which the method is to be applied. The XML provides an easy way of introducing data, making it simple to describe environments for this Path Planning solution.

A.2 Usage Example

Figure A.1 provides an input example which defines an environment with a robot, target goal and two obstacles.

```
 1: <Scenario width="300" height="300">
 2:
 3: <Robot x="270" y="150"/>
 4:
 5: <TargetPoint x="30" y="150"/>
 6:
 7: <ObstacleList>
 8:
 9: <RectangularObstacle x="195" y="220" width="20" height="120"/>
10:
11: <CircularObstacle width="10" height="10" x="70" y="50"/>
12:
13: </ObstacleList>
14:
115: </Scenario>
```

Figure A.1: Potential Fields XML input file example.

Appendix B

Prototype Hardware Specification

B.1 Lego's NXT Brick

Component	Characteristics
Main Processor	Atmel® 32-bit ARM® processor, AT91SAM7S256 - 256 KB FLASH - 64 KB RAM - 48 MHz
Co-Processor	Atmel® 8-bit AVR processor, ATmega48 - 4 KB FLASH - 512 Byte RAM - 8 MHz
Bluetooth Wireless Communication	CSR BlueCoreTM 4 v2.0 + EDR System - Supporting the Serial Port Profile (SPP) - Internal 47 KByte RAM - External 8 MBit FLASH - 26 MHz
USB 2.0 Communication	Full speed port (12Mbits/s)
4 Input Ports	6-wire interface supporting both digital and analog interface - 1 high speed port, IEC 61158 Type 4/EN 50170 compliant
3 Output Ports	6-wire interface supporting input from encoders
Display	100 x 64 pixel LCD black & white graphical display - View area: 26 X 40.6 mm
Loudspeaker	Sound output channel with 8-bit resolution and a sample rate of 2-16 KHz
4 Button User-Interface	Rubber Buttons
Power Source	6 AA Alkaline batteries or a Rechargeable Lithium-Ion battery 1400 mAH
Connector	6-wire industry-standard connector, RJ12 Right side adjustment

Table B.1: Hardware specification for the NXT Brick [Lego, 2006].

B.2 Smartphone Nokia N80 and Nokia N95

Characteristic	Nokia N80	Nokia N95
Operating System	Symbian OS v9.1	Symbian OS v9.2
Developer Platform	Series 60 3rd Edition	Series 60 3rd Edition Feature Pack 1
CPU	CPU Type: ARM 9 CPU Clock Rate: 220 MHz	CPU Type: ARM 11 Clock Rate: 332 MHz 3D Graphics HW Accelerator
Camera	Resolution: 2048 x 1536 Sensor: CMOS 3.0 Megapixels Focal length 4.7 mm F-Stop/Aperture f/3.5 Focus range 17 cm to infinity Digital Zoom: 20 x Image Format: JPEG/Exif Feature: Flash, Red-Eye Reduction, Self Timer Video Resolution: 352 x 288 Video Frame Rate: 15 fps Video Zoom: 5 x Video Format: H.263, MPEG-4	Resolution: 2582 x 1944 Sensor: CMOS 5.0 Megapixels Focal length 5.6 mm F-Stop/Aperture f/2.8 Focus range 10 cm to infinity Digital Zoom: 20 x Image Format: JPEG/Exif Feature: Auto Focus, Carl Zeiss Optics, Flash, Red-Eye Reduction, Self Timer Video Resolution: 640 x 480 Video Frame Rate: 30 fps Video Zoom: 10 x Video Format: H.263, MPEG-4
Memory	Max User Storage: 40 MB NAND Memory: 128 MB SDRAM Memory: 64 MB 18 MB Free Executable RAM Memory Memory Card: Mini SD Max Memory Card Size: 2 GB Memory Card Feature: Voltage: 1.8V, 3V, Hot Swap Unlimited Heap and Jar size within memory bounds	Max User Storage: 160 MB NAND Memory: 256 MB SDRAM Memory: 64 MB 18 MB Free Executable RAM Memory Memory Card: Micro SD Max Memory Card Size: 4 GB Memory Card Feature: Hot Swap Unlimited Heap and Jar size within memory bounds
Bluetooth	v1.2 (1 Mbit/s)	v2.0 with EDR (3 Mbit/s)
Java Support	MIDP 2.0 CLDC 1.1 JSR 135 Mobile Media API JSR 172 Web Services API JSR 177 Security and Trust Services API JSR 179 Location API JSR 180 SIP API JSR 184 Mobile 3D Graphics API JSR 185 JTWI JSR 205 Wireless Messaging API JSR 75 FileConnection and PIM API JSR 82 Bluetooth API	MIDP 2.0 CLDC 1.1 JSR 135 Mobile Media API JSR 172 Web Services API JSR 177 Security and Trust Services API JSR 179 Location API JSR 180 SIP API JSR 184 Mobile 3D Graphics API JSR 185 JTWI JSR 205 Wireless Messaging API JSR 226 Scalable 2D Vector Graphics API JSR 234 Advanced Multimedia Supplements JSR 75 FileConnection and PIM API JSR 82 Bluetooth API

Table B.2: Characteristics comparison between Nokia N80 and Nokia N95.