Path Planning of Robots

Jan Holdgaard-Thomsen

Kongens Lyngby 2010
IMM-M.Sc.-2010-5

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-M.Sc.-2010-5

Summary

Hospitals are ideal candidates for automation of transportation, as many items are transported daily from one location to another. The heavy load of transportation tasks requires a significant amount of manpower, and common tasks are mainly concerning transportation of trash, food, laundry, beds, samples, medical equipment, office supplies, mail etc.

In this thesis, the aim was to analyse this situation in a large, and with a focus on path planning as the central theme of this thesis. The thesis covers a number of ways to characterize the path planning problem and the algorithms used to address it. The path planning strategy for a robot is a complex problem, it involves a wide range of subjects: complexity theory, mapping, path planning and more.

The presented strategies are evaluated based of the requirements for a hospital environment like Bispebjerg Hospital. The thesis specifies the main problems and challenges that need to be considered when designing such systems.

Resumé

Hospitaler er ideelle miljøer til automatisering af transportopgaver, mange ting transporteres dagligt fra et sted til et andet. Den tunge belastning af transportopgaver kræver en betydelig mængde arbejdskraft; opgaverne handler hovedsagelig om transport af skrald, mad, vasketøj, senge, prøver, medicinsk udstyr, kontorartikler, post osv.

I specialet var formålet at analysere denne situation i et større perspektiv, med fokus på ruteplanlægning som det centrale tema. Specialet omfatter en række forskellige måder at karakterisere ruteplanlægnings problemet og de algoritmer, der anvendes til at løse det. Ruteplanlægning for en robot er et komplekst problem, der involverer en bred vifte af emner: kompleksitetsteori, kortlægning, ruteplanlægning mm.

De præsenterede strategier er vurderet på grundlag af de krav som stilles til et hospitalmiljø som Bispebjerg Hospital. Specialet angiver de vigtigste problemer og udfordringer, som skal tages i betragtning ved udformningen af sådanne systemer.

Preface

This thesis was prepared at the Department of Informatics and Mathematical Modelling, the Technical University of Denmark in fulfilment of the requirements for acquiring the M.Sc. degree in engineering.

The thesis was written in the period from the 1th of September 2009 to the 22th of February 2010. A current research project involving DTU Informatics, DTU Management, Force Technology, and Bispebjerg Hospital aims to reduce the required manpower by letting these transportation tasks be carried out by a mobile robot.

I would like to thank Thomas Bolander for consistently providing support and input to the process while ensuring that I kept moving forward. I would also like to thank the individuals in the research project that have influenced the work.

I have written this thesis assuming that the reader comes with certain knowledge about algorithms in the field of path planning, and knowledge of basic mathematical concepts.

Lyngby, February 2010

Jan Holdgaard-Thomsen

Contents

Chapter 1

Introduction

Automation techniques are applied widely in many fields like industrial plants, factories and offices. The main purpose of automation is both to save time and manpower and to improve the service quality.

Hospitals are ideal candidates for automation of transportation, as many items are transported daily from one location to another by trained people. The heavy load of transportation tasks requires a significant amount of manpower, and common tasks are mainly concerning transportation of trash, food, laundry, beds, samples, medical equipment, office supplies, mail etc. This is not only waste of the trained personnels expertise but also a monotonous and tedious work.

Automation is a key aspect for increasing efficiency and saving both time and manpower. The main purpose of an automatic technology is to reduce the need of human transportation tasks and replace them with robotic technology where the transportation task is suitable. A mobile robot that can perform transportation tasks could therefore relieve hospital personnel of this time consuming, secondary task and free their time for the more critical primary duties

The solution will require an automation system consisting of a mobile robot there is able to transport the different hospital equipment from one place to another. The robot must be suitable for hospital environment where human traffic will not interfere with the system and not lead to potentially collisions with people or interior. The system must therefore be able to deal with changes in its surrounding environments such as people and other obstacles.

One technology that has been successfully implemented in many industrial applications to move materials around a manufacturing facility of a warehouse is Automatic Guided Vehicle AGV. An AGV is a mobile robot which means the robot is not fixed to one physical location. The benefit of robotic technology and the increasing knowledge from the research and the experience in industrial environments, result in introducing solutions to transportation of goods in places like hospitals. There exist several examples on utilizing robotic technology in hospitals. One of the earliest service robot projects is the HelpMate [19] robot capable of navigating hallways and riding elevators to deliver payloads to programmable destinations throughout a hospital. TUG [14] is another commercially available robot, which is able to transport attached wagons by tugging them. Another motivating example is the mobile robot named RHINO [15] which is a fully autonomous tour-guide at the Deutsches Museum Bonn. RHINO is able to lead museum visitors from one exhibit to the next by calculating a path using a stored map of the museum.

These examples indicate an increasing interest for such robotic systems. The project about implanting robots in a hospital was started by a Danish hospital, a company located in Copenhagen and Technical University of Denmark. This project is concerning the sub problem of *Path Planning* this involves a number of non-trivial subjects; this thesis will point out some of the important concepts in path planning for mobile robots.

1.1 Hospital Environment

This section describes the environment of the hospital where the robot system is to be implemented.

In order to identify the major problems it is necessary to analyse and examine the needs of a transportation system. The case is Bispebjerg Hospital located in Copenhagen and consists of 51 buildings placed in a green oasis in the city. The hospital has a capacity of approximately 500 beds and 3000 full time employees. Bispebjerg Hospital has gained status as a field hospital and the hospital is gradually growing to serve 400000 people. The hospital is built around the original pavilions from 1913 and a few newer buildings, all connected through an underground tunnel system. Today the majority of the transportation tasks use the tunnel system. The tunnel network has a total length of 5 km, and the longest distance from one point to another is 1.5 km. Elevators combined with the tunnel system are suitable for transporting daily hospital equipment like beds, garbage and food containers [29] *(pp. 392-397)*. The environment at the hospital is rarely changing but furniture may be rearranged or hospitals beds are placed in the hallway. And certain hallways are crowded at some specific time of the day. To realize mobile service robots other requirements need to be considered. They include ability to work without any adjustment to the environment for feasible implementation. To function in large scale workspace for comprehensive working range.

1.1.1 Mobil Robot

The mobile robot technology provides a solution to the hospital transportation problem. The mobile robot must be able to deal with changes in its surrounding environment; a mobile robot does not move along a fixed track and is therefore more suitable for environments where humans are involved. The development of a mobile robot requires integration of capabilities, such as localization, mapping, path planning, navigation, visual tracking, planning, signal processing and human robot interface etc. This project is avoiding reinventing what is made available by others and focusing efforts towards new discoveries. Designing a software environment for robotic systems is not an easy task. Software development is a necessary process in autonomous mobile robotics and is becoming more and more important to assist developers in their scientific and engineering work. Many existing programming environments, like The Player Project [45] *(pages 2421-2427)*, Carmen [23] *(pp. 2436-2441)*, CLARaty [25] *(pp. 2428-2435)*, OROCOS [28] *(Vol. 14, pp. 33-49)* and SmartSOFT [36] *(pp. 253-263)* are all proposing different approaches for Robotics Development Environments (RDE). Most of them are different in the use of specific communication protocols and/or mechanisms, different operating system, robotics platform, programming language etc. This project will not describe the different RDE since the RDE for this project has been chosen to be Carmen [23] because it supports the hardware of the mobile robot a Pioneer 3-AT from *http://www.mobilerobots.com/*.

1.2 Preconditions

This purpose of this section is to clarify the authors knowledge about the problem, the decision of the mobile robot control software environment is already chosen to be Carmen [23]. This thesis will therefore not include an analysis and evaluation of various robotics software environments.

1.2.1 Robotic Properties

In preparing of this thesis many properties of the robot are still unknown, effective motion planning depends heavily on properties of the robot solving the task. For example, if the robot is equipped with a car-like steering it can theoretically reach any point and achieve any orientation within its two dimensional environment, although it might have to carry out several and varied manoeuvres to do so. This is due to the fact that this type of robot is only capable of executing a single displacement either forward or backward and a limited steering angle. This fact also means that this type of robot is capable of only two controlled Degrees of Freedom (DOF) which is one fewer than are available in its two dimensional environment. The robot is therefore non-holonomic. In general, a placement of a robot is specified by the number of parameters that corresponds to the number of DOF of the robot. This number is two for planar car-like steering robots, and its three for planar robots that can rotate as well. The geometry of car-like is more complicated and since the properties of the actual robot are unknown I find it fair to restrict the robot of this thesis to be able to move in arbitrary directions. This thesis defines a method to represent a robot by a point in a coordinate system, by definition this point is then called the *reference point* of the robot.

1.2.2 Requirements

Performance requirements for the robot are not yet decided, therefore, the criteria listed in this thesis are decided by the author. Since we deal with robots that carry out transport tasks, this thesis will mainly concentrate on complete planners. Complete planning algorithms must find a path for the robot whenever one exist and may only report failure if no path exist, but there is no claims about the quality of the path. The robot could make a large detour, or make lots of unnecessary turns. In practical situations a good path is preferred not just any path. What constitutes a good path depends on the robot. In general, the longer a path, the more time it will take the robot to reach its goal position. For a mobile robot this means it can transport fewer goods per time unit, resulting in a loss of productivity. Therefore I will prefer a short path. Often there are other issues that play a role as well. For example, some robots can only move in a straight line; they have to slow down, stop, and rotate before they can move into different direction, so any turn along the path causes some delay. For this type of robots not only the path length but also the number of turns along the path has to be taken into account. In this thesis I ignore the aspect of the time it takes the robot move along a specific path. When this thesis referees to the shortest path it means the Euclidean shortest path for a robot.

1.3 Computational Complexity

The objective of robot motion planning is to give the robot the capability of planning its own motion in an environment filled with obstacles. Path planning is a problem-solving competence, as the robot must decide what to do over the long term to achieve its goal. Different types of motion planning problems may differ in their "size" e.g. the dimension of the environment and the number of obstacles. It is important to quantify the performance of the planning methods, typically the time required to solve the problem and the computational complexity of the planning problem. Analysis of the methods is useful to judge their practicality and to detect opportunities for improvements. The analysis of the problem can also be useful to suggest new ways of formulating problems when the original formulation is too costly.

1.3.1 PSPACE-hard problem

This thesis will review an important theoretical result concerning the computational complexity of robot motion planning. The problem was established by Reif [31] for planning free paths in a configuration space of arbitrary dimension, often referred to as the "Generalized Mover's Problem" [31] or "Piano Movers' Problem" [38].

> *Planning a free path for a robot made of an arbitrary number of polyhedral bodies connected together at some joint vertices, among a finite set of polyhedral obstacles, between any two given configurations, is a PSPACE-hard problem [20].*

This regards the complexity of computational problems in terms of the amount of space, or memory, that they require. Time and space are two of the most important considerations when seeking practical solutions to many computational problems [42]. Explaining the definition of PSPACE will make use of the Turing machine model [42]. Turing machines are mathematically simple and close enough to simulate real computers to give meaningful results. PSPACE is the class of problems that are decidable in polynomial space on a deterministic Turing machine. Additional study on the exact path planning techniques for the generalized mover's problem led Schwartz and Sharir [37] to an algorithm that is expected to require at least exponential time on a nondeterministic Turing machine, hence double exponential time on a deterministic Turing machine [20] [10].

The complexity of path planning described in the literature and in this section gives strong evidence that the time and space required to solve a motion planning problem increases rapidly with the dimension of the environment. Therefore it is important to simplify the problem but still retain the problem realistically. If the problem is reasonably well understood and if it is acceptable to trade some generality against better performance such a simplification is almost always feasible [20]. The simplification of the problem can be performed by the user or automatically generated by an algorithm from the original input.

The following chapters present techniques for simplifying the motion planning problem. Most of the discussion focuses on the robot representation, the representation of the environment and how to map the environment afterwards its dealing with path planning in the representation. After that it is time to analysis the methods used by Carmen and verify if Carmen is able to deliver complete path planning and provide shortest path for a robot.

Chapter 2

Configuration Space

To create motion plans for robots, there is a need of specifying the dimensions of the robot and obstacles in the environment. More specifically, there must be given a specification of the location of every point on the robot and each obstacle, since the need of ensuring that no point on the robot collides with an obstacle. In this chapter the Configuration Space (C-space) [20] [10] is introduced.

The purpose is to give the robot the capability of planning its own motion, deciding automatically what motions to execute in order to achieve a task specified by initial and goal position. Before the robot is able to plan a motion the robot must include knowledge about the environment in which it is moving. For example mobile robots moving around in hospitals must know where obstacles like walls are located. The robot should be able to detect obstacles that are not included on the map like people moving around. For that information the robot will have to rely on its sensors. Using information about the environment the basic motion planning problem is defined as to produce a continuous motion in the C-space [20] [10] [11] connecting a start and a goal configuration while avoiding obstacles in the environment.

The goal of defining a basic motion planning is to isolate some central issues and analysis them in depth before considering additional difficulties. This section introduces some of the basic notions used in motion planning. The general motion planning is complex task and there will be a number of simplifying assumptions.

2.1 Robot Representation

Let $\mathcal{R}$ be a robot moving around in a two dimensional environment. The environment will be a planar region with polygonal obstacles, since the actual size and shape of the robot in unknown I assume that $\mathcal{R}$ is a simple polygonal. The robot motion depends on its mechanics. Some types of robots can move in any direction, while others are limited motion. Robots with four wheels mounted like a car cannot move sideways, and they often have a certain minimum turning radius. Other robots can changes their orientation by rotation. C-space describes the space of possible positions the robot may attain. For example in a 2-dimensional environment a placement or configuration of $\mathcal{R}$ can be specified by a translation vector. For instance, if the robot is a polygon with vertices $(1, -1)$, $(1, 1)$, $(0, 3)$, $(-1, 1)$ and $(-1, -1)$, then the vertices of $\mathcal{R}$ $(6, 4)$ are $(7, 3)$, $(7, 5)$, $(6, 7)$, $(5, 5)$ and $(5, 3)$. With this notation, a robot can be specified by listing the vertices of $\mathcal{R}$.

An alternative way to view this in terms of a point placed in the center of the robot. This point is then called the reference point of the robot. Then a placement of the robot can be specified by simply stating the coordinates of the reference point. Suppose the robot can change its orientation by rotation around its reference point. Then an extra parameter is needed to specify the orientation of the robot. Let $\mathcal{R}$ (x, y, ϕ) denote the robot with its reference point at (x, y) and rotated though an angle ϕ as illustrated in Figure 2.1.

In general, a placement of a robot is specified by the number of parameters that corresponds to the number of *Degrees of Freedom* of the robot. This number is three for planar robots able to rotate as well as move. The C-space of a moving robot in the plane is identical to the work space. Still, it is important to distinguish the two notions. The work space is the space where the robot actually moves around and the C-space is the parameter space of the robot. A polygonal robot in the work space is represented by a point in C-space, and any point in C-space corresponds to some placement of a robot in work space.

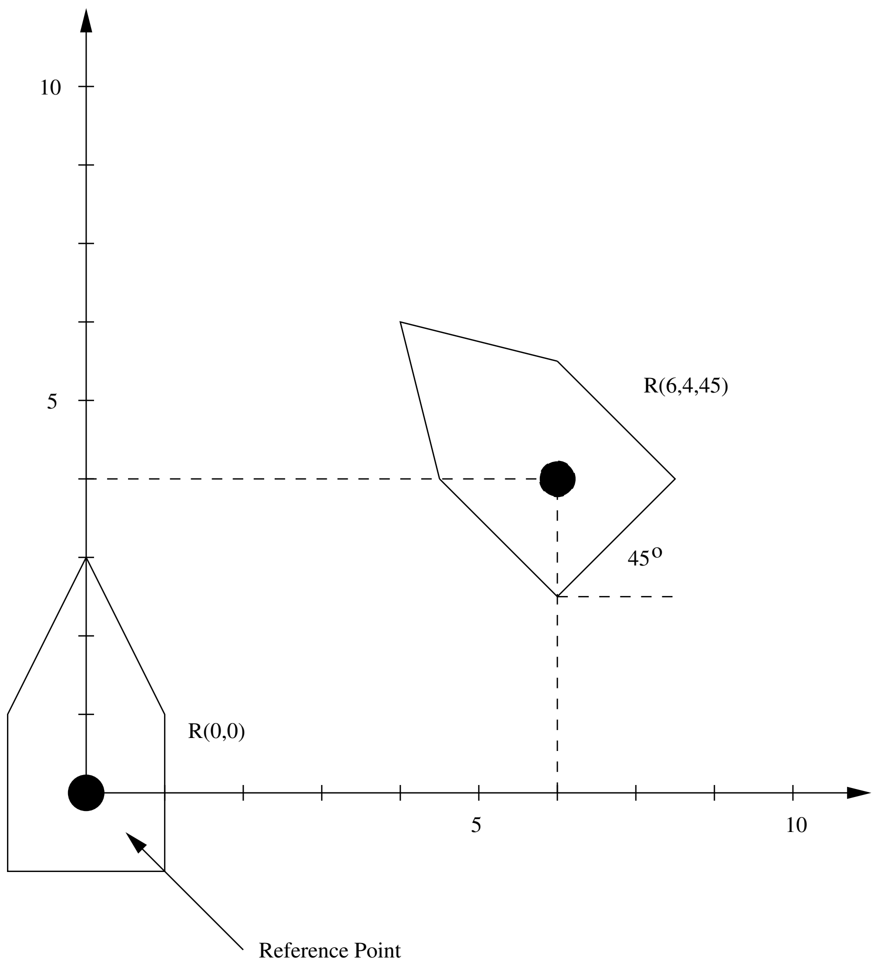

Figure 2.1: Describing the placement of a robot with three parameters, placement and rotation.

2.2 Obstacle Representation

It has now been described how to map placements of the robot to points in the C-space. The next thing to examine is how obstacles in the workspace can be mapped into the C-space for the robot. An obstacle $\mathcal{P}$ is mapped to the set of points p in C-space such that $\mathcal{R}(p)$ intersects $\mathcal{P}$. Consider a circular mobile robot in an environment with a single polygonal obstacle in the workspace, as shown in Figure 2.2. In Figure 2.2(b) the robot is moving around the obstacle and keeping track of the curve traced out by the reference point. The reference point is chosen to be placed in the center of the robot, but could easily be placed at another position. Figure 2.2(c) shows the resulting obstacle in the C-space. Motion planning for a circular robot in Figure 2.2(a) is now equivalent to motion planning for a point in the C-space, shown in Figure 2.2(c). Any open space not occupied by an object is free space, where the robot is free to move without hitting anything modeled.

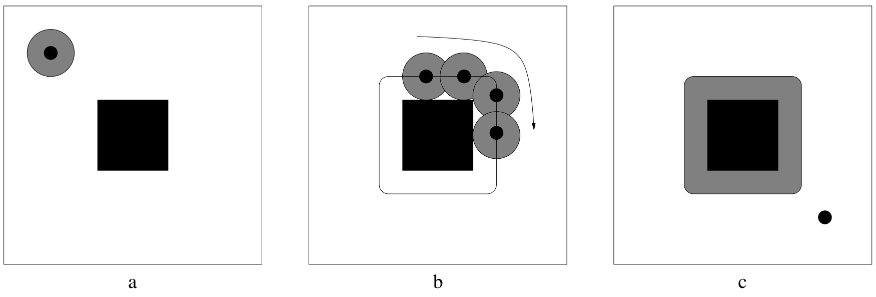

Figure 2.2: (a) The circular robot approaches the workspace obstacle. (b) By moving the robot around the obstacle the Configuration Obstacle Space (C-obs) is created. (c) Motion planning in the workspace representation (a) has been transformed into motion planning in C-space.

The question that remains to be answered: is the robot colliding with an obstacle if it touches that obstacle? The obstacles can be defined to be topological open or closed [11]. Obstacles defined as open sets mean that the robot is allowed to touch them. In practice a movement where the robot passes very close to an obstacle cannot be considered safe because of possible errors in robot control. Such situations can be avoided by slightly enlarging all the obstacles before computing a path.

The overall idea of configuration space formulation is to represent the robot as a point in the environment and to map the obstacles. The set of configurations that avoids collision with obstacles is called the Configuration Free Space (C-free). The complement of C-free in C-space is called the C-obs or forbidden region. It was recognized early on in motion planning research that the C-space is a useful way to abstract planning problems in a unified way. The advantage of this abstraction is that a robot with a complex geometric shape is mapped to a single point in the C-space. For path planning purposes the robot can be modelled as a point as in Figure 2.2 so that the orientation is unimportant. This simplicity assumes the robot is holonomic, or can turn in place. Many research robots are sufficiently close to be holonomic to meet this assumption like tour-guide robots that are able to interact and present exhibitions [40] [7]

[27]*(pp. 95-124)* [39]*(pp. 203-22).*

The concept of metric path planning converts the representation of the environment to a C-space representation that facilitates path planning. There exist a large number of different C-space representations. The representations offer different ways of partitioning free space. Any open space not occupied by an object is free space, where the robot is free to move without hitting anything obstacles. Each partition can have additional information like "this area is off-limit from 9am to 5am," etc.

The next step will be constructing a data structure for storing free space. This data structure can then be used to compute a path between any two given initial and goal positions. This approach is useful when the work space is rarely changed and many paths have to be computed. The possibilities to compute a representation of the free space will be discussed in the next chapter.

Chapter 3

Environment Representation

The environment representation can make a huge difference in the performance and path quality. This chapter introduces and discusses the metric and topological methods for building maps.

Building a representation of the environment is an important task for a mobile robot that aims to move autonomously in the surrounding environment. If many paths were to be planned in the same environment it would make sense to construct a data structure once and then use that data structure to plan subsequent paths more efficiently. This data structure is often called *a map* and *mapping* is the problem of exploring and sensing an unknown environment to construct a representation that is useful for navigation.

The literature of robotics describes two common paradigms for modelling indoor environments: *metric* or *grid-based* paradigm and *topological* paradigm [46].

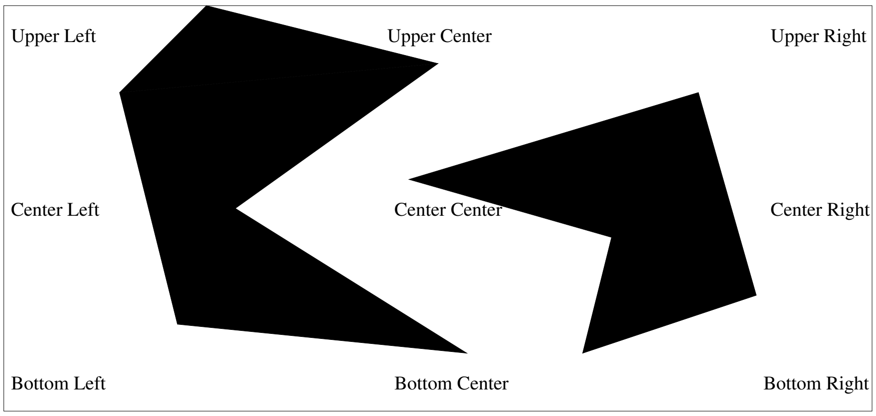

Figure 3.1: A real-world environment, areas shaded black represent obstacles in the environment white areas represent free space.

3.1 Topological Approach

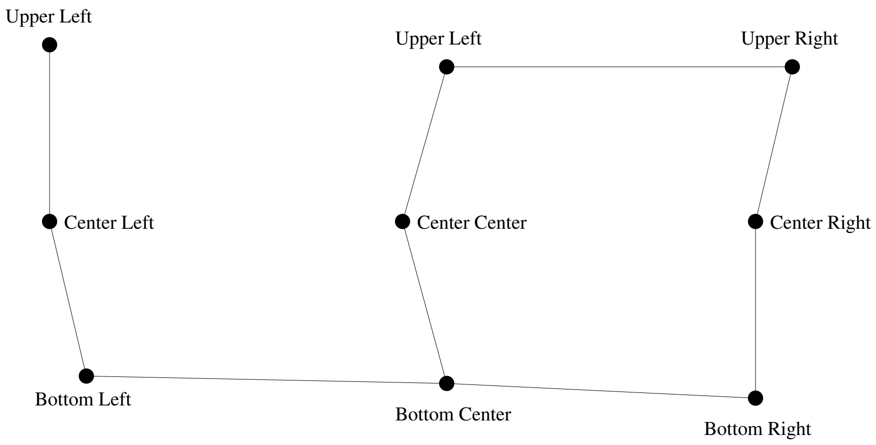

Figure 3.2: A topological map of the same area as Figure 3.1, with vertices representing places such as regions, and edges representing navigable paths between these places.

Topological representations aim at representing environments with a graph structure shown in Figure 3.2 where vertices correspond to distinct locations or landmarks. They are connected by an edge if there is a direct path between them. For example locations can be intersections and T-junctions in a building or an office. Figure 3.1 shows a real-world representation.

3.2 Metric Approach

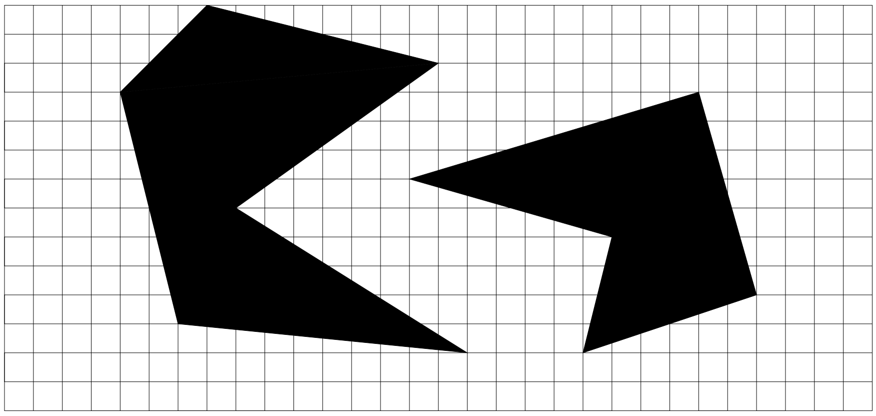

Figure 3.3: A metric map of Figure 3.1 with seven rooms connected by doors. Space is divided into equally sized squares with obstacles shaded black and free spaces blank.

A metric (grid-based) map represents environments by regular two dimensional grids shown in Figure3.3 and each grid cell may indicate the presence of an obstacle or free space in the corresponding region of the environment. In these representations the environment is defined by a single global coordinate system, in which all mapping and navigation takes place. Regular grids are often referred to as *occupancy grids* [24].

3.3 Evaluation of Both Paradigms

Both approaches to mapping the environment either topological or metric have significant strengths and weaknesses. Metric maps are simple to construct and maintain even in large-scale environments [6]. The grid-based approaches distinguish between different places based on the geometric position within a global coordinate system. The position of the robot is estimated based on odometric information and sensor readings from the robot. Thus occupancy grid approaches also use a number of sensor readings to determine the location of the robot. To the extent that the position can be tracked accurately different positions for which sensor readings look alike are disambiguated by odometric information. Geometric locations are recognized as such even if the sensor readings are different which is often the case in dynamic environments where movable interior, open doors or people can block the sensor of the robot. Grid-based approaches suffer from their time and space complexity since the resolution of the grid must be dense enough to capture all important details from the environment.

The topological paradigm has a different approach of determining the position of the mobile robot this is based on landmarks detection. A topological map represents the robot environment by a graph. Paths are defined as sets of two distinct vertices or locations which must be detected and recognized by the robot using sensors data. These points provide the vertices of the map. Vertices correspond to locations where special actions need to take place like turning, entering a door, launching another navigation behaviour etc. Locations are required to have some unique distinguishing feature that can be reliably recognized by a robot. Only a few relevant information about the locations are required to locate and identify them [35].
The edges between two vertices correspond to trajectory segments where navigation operations can be used such as wall following, visual servoing, etc. These navigation operations take the robot from one vertex to another. In the example where the robot traverses two places that have a similar look topological approach often have difficult determining if these places are identical. Particular if these places have been reached via different paths [44]. Since sensor input depends strongly on the view-point of the robot topological approaches may fail to recognize geometrically places. For example in situations where the robot travels over a bump e.g. a cable lying on the floor are difficult to model and can lead to unpredictable results.

Some of the advantages by the topological approach is the path between two vertices does not have to be traced exactly it is sufficient if the robot can traverse a general path not exactly defined between two vertices. This means there is only required small storage capacities. This makes the topological approach to a convenient representation for problem solvers. They also provide a more natural interface to human instructions: like "go to office 12". Since topological approaches do not require exact determination of the geometric position of the robot, this approach usually recover better from slippery regions where odometric information gets inexact. This phenomenon must constantly be monitored and compensated in grid-based approaches. Moreover it is easier to generate and maintain global consistency for topological approaches than for grid-based. Since topological approaches usually do not require exact determination of the geometric position of the robot.

However two important properties have to be guaranteed: first the vertices have to be detected and identified with certainty and accuracy and secondly the navigation operations must lead the robot from one vertex to another. The following chapters 4, 5 and 6 discuss three general strategies for decomposition of the C-free.

- **Roadmap** This approach is based on the following idea of capture the connectivity of the robots C-free in the form of a network of one dimensional line segments lying in C-space. Once a roadmap is constructed the roadmap can be used as a set of paths. Path planning is then reduced to connect the initial and the goal state and searching the roadmap for a path [20]

- **Cell Decomposition** This approach is to decompose the C-free into a collection of non-overlapping regions called cells, whose union is exactly C-free. Next thing is to construct a *connectivity graph* which represents the adjacency relation among the cells which facilitates path searching. If successful the output of the search is a sequence of cells often referred to as a *channel* [20], connecting the cells including the initial and the goal state.

- **Potential Field** The two previous described methods aim at capturing the connectivity of the C-free into a graph that is subsequently searched for a path. The potential field method is based on a different idea. This approach creates a field or gradient across the C-free that directs the robot to the goal position from multiple prior positions.

The following sections present common instantiations of the road map, cell decomposition and potential fields. The sections introduce these approaches and illustrate them with simple examples in two-dimensional configuration space populated by polygonal obstacles.

Chapter 4

Roadmaps

This section focuses on the class of topological maps called roadmaps. The roadmap is a decomposition of the robot configuration space based specifically on obstacle geometry. The challenge is to construct a set of paths that together enables the robot to go anywhere in its free space while minimizing the number of total roads. A roadmap is embedded in the free space and the vertices and edges is included with physical meaning. Roadmap vertices correspond to a specific location and an edge correspond to a path between neighbouring locations. The roadmap approach is based on the general idea of capture the connectivity of the robots C-free in the form of a network of one-dimensional lines that captures the topology of the free space [20]. Once a roadmap is constructed, it is used as a network of road segments for robot motion planning. Robots make use of roadmaps in much the same way as people use the highway system. Instead of planning every single street to a destination people normally plan their path to a network of highways, along the highway and finally from the highway system to their destination. The key issue in this approach is obviously the construction of the roadmap. Various methods based on this general idea have been proposed. They compute different types of roadmaps called *visibility graphs* and *Voronoi diagrams.* In the case of visibility graphs roads become as close as possible to the obstacles and results in paths that are minimum-length solutions. Contrasting with the visibility graph approach a Voronoi diagram is a complete roadmap method that tends to maximize the distance between the robot and obstacles on the map. Therefore the path in the Voronoi diagram is usually far from optimal in the sense of shortest path [40]. The Voronoi diagram method has an important weakness in the case of limited range sensors. Since the method maximizes the distance between the robot and obstacles in the environment any short range sensor will have the risk of failing to sense its surroundings. On the other hand the visibility graph method can be designed to keep the robot as close as desired to obstacles therefore this thesis will only concentrate on the visibility graph approach.

4.1 Visibility Graphs

The characteristics of a visibility graph are that its vertices share an edge if they are within line of sight of each other and that all points in the robots free space are within line of sight of at least one vertex on the visibility graph [10] as shown in Figure 4.1. This method is one of the earliest path planning methods

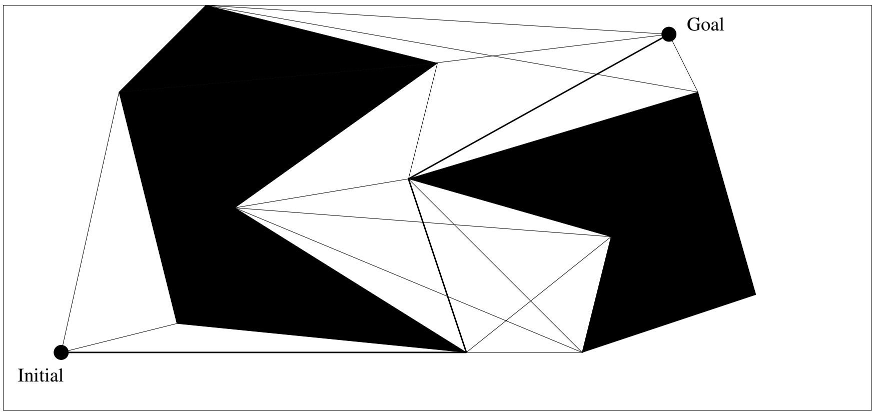

Figure 4.1: The thin lines indicate the edges of the visibility graph of two obstacles represented as filled polygons. The thick line represents the shortest path between the initial and goal position.

and has been widely used to implement path planners for mobile robots [20]. There are however two important cautions when using visibility graphs search:

- The size of the representation and the number of the edges and vertices increases with the number of obstacles polygons. This method can therefore be slow and inefficient compared to other methods when used in environments with a compact existence of obstacles. On the other hand can the method be extremely fast and efficient in sparse environments.

- The solution path found by visibility graph planning tend to take the robot as close as possible to obstacles when moving around in the environment. It can be proved that visibility graph is optimal in terms of the length of the solution path [20] [40]. This result means that the safety in terms of staying at reasonable distance from obstacles is sacrificed of the optimal result. One solution could be to increase the size of obstacles by significant more than the radius of the robot, or alternative to modify the solution path after planning to distance the path from obstacles when possible. Of course these modifications will affect the optimal-length result of visibility graph path planning.

4.1.1 Computing the Visibility Graph

Let $V = v_1, ..., v_n$ be the set of vertices of the polygons in the C-free as well as the initial and goal position. The construction of the visibility graph for each $v \in V$ must determine which other vertices are visible to v. One obvious method to make this determination is to test all line segments $v\vec{v}_i$, $v \neq v_i$ to verify if they intersect an edge of any polygon. For a particular line segment $v\vec{v}_i$, there are $O(n)$ intersections to check because there are $O(n)$ edges from the obstacles. There are $O(n)$ potential segments emanating from v. So for a particular v there are $O(n^2)$ tests to determine which vertices are visible from v. This must be done for all $v \in V$ and therefore the computation of the visibility graph would have complexity $O(n^3)$. This algorithm can be improved by using a variant of the sweep line algorithm [20] and [10]. This algorithm allows to construct the visibility graph in $O(n^2\ log\ n)$. A plane sweep algorithm solves the problem by sweeping a line across the plane and makes a stop at each of the vertices of the obstacles. At each vertex, the algorithm updates a partial solution to the problem. The purpose of plane sweep algorithm is to efficiently compute the intersections of a set of line segments in the plane to compute intersections of polygons.

The solution paths found by the visibility graph tend to take the robot as close as possible to obstacles. Therefore it can also be proved that visibility graph is optimal in therms of length of the solution path [40] For a shortest path to exist it is important that obstacles are open sets. If obstacles were closed then the shortest path would not exist unless the robot is able to move to its goal in a straight line. Even if the obstacles are closed sets it would always be possible to shorten a path by moving closer to an obstacle until collision. This optimal result also means that some sense of safety in terms of staying in a reasonable distance from obstacles is sacrificed for this optimality.

The common solution is to enlarge obstacles by significant more than the robots radius, or alternative, to modify the path to distance from obstacles when possible. Such actions sacrifice the optimal length results of visibility graph path planning.

CHAPTER 5

Cell Decompositions

The idea behind cell decomposition is to decomposing the C-free into simple regions called cells. A non-directed graph represents the adjacency relation between the cells. The shared boundaries of cells often have a physical meaning such as a change in the obstacle or a change in line of sight to surrounding. Two cells are adjacent if they share a common boundary a non-directed adjacency graph encodes the adjacency relationships of the cells. From this graph a continuous path or channel can be determined by simply following adjacent free cells from the initial point to the goal point. Cell decomposition methods can be broken down into *exact* and *approximate* methods:

- **Exact Cell Decomposition** The first step in the method of cell decomposition is to decompose the C-free which is bounded both internally and externally by polygons into trapezoidal and triangular cells by simply drawing parallel line segments from each vertex of each interior polygon in the C-space to the exterior boundary. Then each cell is numbered and represented as a vertex in the adjacency graph. Vertices that are adjacent in the C-space are linked in the adjacency graph. A path in this graph corresponds to a channel in free space which is illustrated by the sequence of striped cells. This channel is then translated into a free path by connecting the initial position to the goal position through the midpoints of the intersections of the adjacent cells in the channel.

- **Approximate Cell Decomposition** This approach is different because it uses a recursive method to continue subdividing the cells until one of the following situations occurs. Each cell lies either completely in C-obs region or completely in C-free. The other situation is when an arbitrary limit in resolution is reached. Once a cell fulfils one of these criteria, it stops decomposing. In some literature this method is often called *quadtree decomposition* because a cell is divided into four smaller cells of same shape each time its decomposed.

Both approaches for cell decomposition have advantages and disadvantages. Provided that each method is equipped with both appropriate search techniques and exact computation techniques, exact cells decomposition method are complete meaning they are guaranteed to find a free path whenever one exists or return failure otherwise. Approximate methods may not be complete but the precision of the approximation can be made arbitrarily small. However the trade-off for this accuracy is a more complex mathematical process results in a more expense running time.

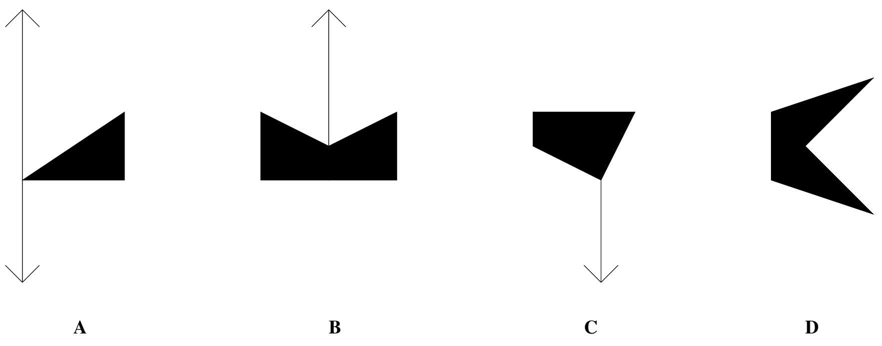

Figure 5.1: (A) Extending upward and downward. (B) Upward only. (C) Download only. (D) Not possible to draw from the vertex in-between the two acute angles.

5.0.2 Computing the Exact Vertical Cell Decomposition

This section will present the principles in constructing a *vertical cell decomposition* which is an exact cell decomposition which partitions C-free into a finite collection of two dimensional cells. Each cell is either a trapezoid or triangle which can be viewed as trapezoids where one of the parallel sides has a zero-length edge. For this reason this method is often called *trapezoidal decomposition.* The decomposition is defined as follows. Let P denote the set of vertices used to define C-obs. At every $p \in P$ draw a *upward* vertical line and one *downward* vertical line through C-free through C-free until C-obs or the boundary of C-space is reached. There are four possible cases, as shown in Figure 5.1 depending on whether or not it is possible to draw a line in the two directions. If C-free is partitioned according to these rules, then the result is a vertical decomposition of C-free shown in Figure 5.2. For simplicity assume that

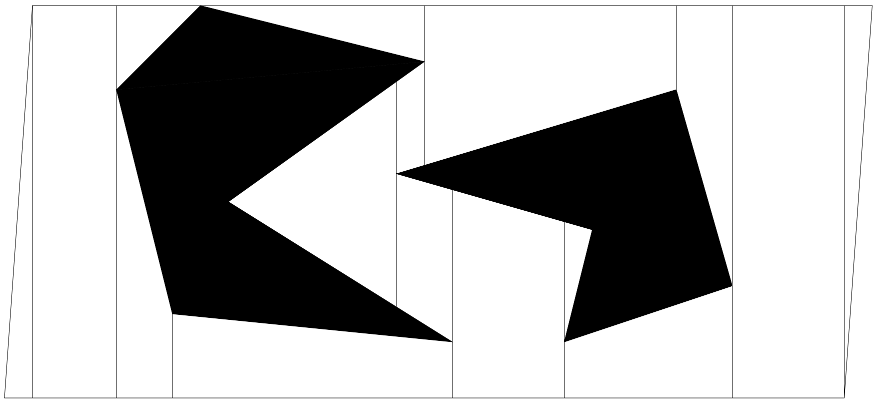

Figure 5.2: The vertical decomposition method uses the cells to construct a roadmap, which is searched to yield a solution to a query.

each vertex on all of the polygons has a unique x-coordinate. A consequence of this is that there cannot be any vertical edges of C-obs. This assumption is not very realistic. Vertical edges occur frequently in many environments and the situation that two non-intersecting obstacles have an endpoint with the same x-coordinate is not unusual either because the precision or resolution (in which the coordinates are given) is often limited.

To deal with this problem a rotation of the axis-system could be a solution. If the rotation angle is small enough then no two distinct will lie on the same vertical line any more. Rotations by very small angles however, lead to numerical difficulties. Even if the input coordinates are integer there is needed a significantly higher precision to perform the calculations properly. It will be convenient not to use rotation, but to use an affine mapping called *shear transformation* [30] Figure 5.3 illustrates the effect. The transformation in which all points along a given line x-axis remain fixed while other points are shifted parallel to the x-axis, this transformation does not change the area of the environment [13].

For computing the decomposition the naive approach appears simple enough that all required steps can be computed by a brute-force algorithm. If C-obs

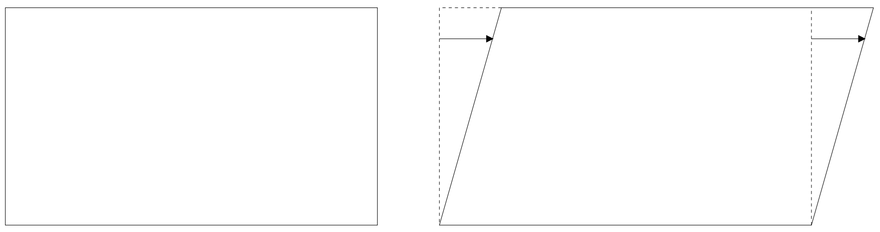

Figure 5.3: A transformation in which all points along a given line x-axis remain fixed while other points are shifted parallel to x-axis.

has n vertices then this approach would take at least $O(n^2)$ time because the intersection test has to be made between each vertical line and each obstacle. This running time can be improved and the resulting running time is only $O(n\ log\ n)$ by making use of the *line-sweep* principle [5] [11]. This principle forms the basis of many combinatorial motion planning algorithms much of computational geometry can be considered as the development of data structures and algorithms that generalize the sorting problem. In others words the algorithms carefully "sort" geometric informations [21]. In principle the algorithm sweeps a line across the plane only to stop where some critical change occurs in the information. To construct the vertical decomposition, imagine that a vertical line sweeps the plane from $x = -n$ to $x = n$, using (x, y) to denote a point in the plane. Note that the only data that appears in the problem input is the vertices and edges of C-obs. It is therefore reasonable that interesting things can only happen at these vertices. Sort the vertices in the input in increasing order by their x-coordinate. Assuming that not two vertices have the same x-coordinate. The vertices will now be visited in order of increasing x value. Each visit to a vertex will be referred to as an *event*. Before after and in between every event a list, L, of some C-obs edges will be maintained. This list must be maintained at all times in the order that the edges appear when intersects by the vertical sweep line.

To be able to handle motion planning queries, a roadmap is constructed from the vertical cell decomposition as shown in Figure 5.4. For each cell C_i let p_i denote a point such that $p_i \in C_i$. The point can be selected as the geometric center. Let $G(V, E)$ be a topological graph defined as follows. For each cell C_i define a vertex $p_i \in V$. There is a vertex for every line and for every cell. For each cell, define an edge from the cells geometric center to the center of every line that lies along each cells boundaries. The resulting graph is a roadmap as in Figure 5.4. The accessibility condition from one point to another point is satisfied because every point can be reached by a straight line path because of the convexity of every cell. Once the roadmap is constructed, the cell information is no longer needed for answering planning queries.

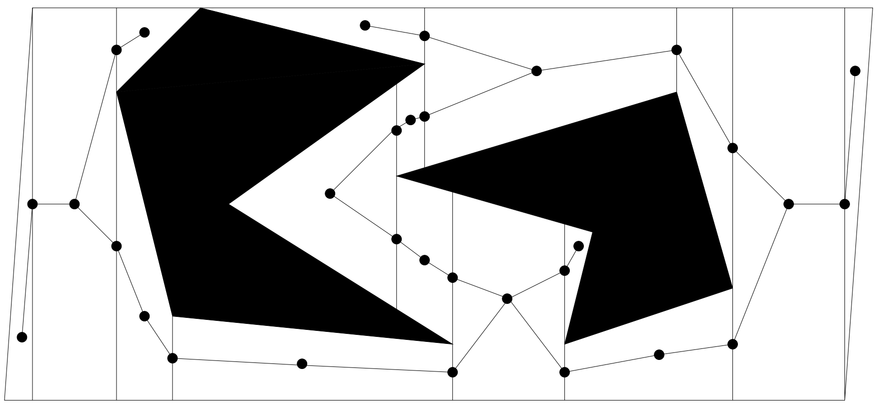

Figure 5.4: The graph derived from the vertical cell decomposition.

Chapter 6

Potential Field

The potential field method treats the robot like a particle moving in a field under the influence of an artificial potential field whose local variations are expected to reflect the structure of the C-free. The robot then moves in the direction indicated by the potential field. The goal acts as an attractive force on the robot also obstacles form a repulsive force directing the robot away from obstacles. The correct combination of repulsive and attractive forces moves robot from the initial position to the goal position while avoiding obstacles. If new obstacles appear during robot motion the potential field needs to be updated in order to integrate this new information.

The drawback of the potential field method is that there is a tendency that the robot might get trapped in local minima which is not the goal position. This can happen when the forces associated with the potentials evaluates to zero and therefore the robot does not move. This happens in the situations where the attractive force towards the goal and the repulsive force are equal. As a result the robot is trapped in local minima and in practice the robot oscillates close to the local minima. This means that there is no guarantee that potential field method will find a path to goal position. In Figure 6.1 the robot initially attracted to the goal as it approaches the horseshoe shaped obstacle. The goal is attracting the robot but the bottom of the obstacle repulses the robot upward until the top of the horseshoe begins to influence the robot. At this point it happens that the attractive force to the goal is symmetric to the repulsive force due to the obstacle surfaces, this leading to a local minimum different then the goal position.

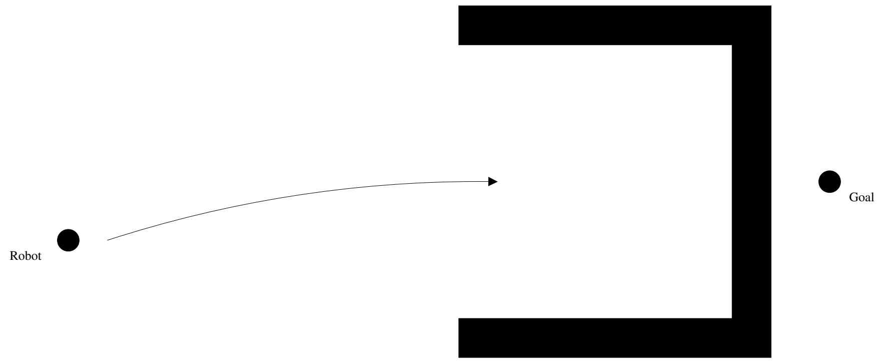

Figure 6.1: Local minima due to concave obstacle.

This problem is not limited to concave obstacles attraction to local minimum also arises with non-concave obstacles as represented in Figure 6.2 where the total repulsive force due to the two obstacles is symmetric to the attractive force due to the goal.

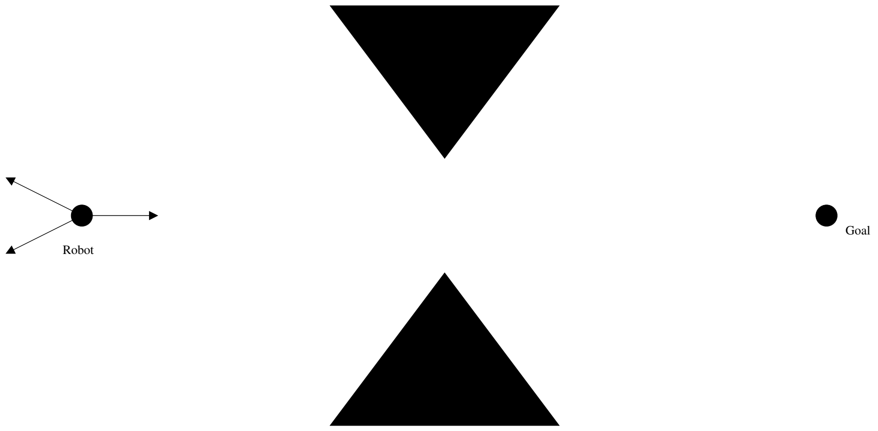

Figure 6.2: Local minima due to environment symmetry.

A large number of improvements and extensions of the potential field have been published since the early version by O. Khatib [16]. Most of them make use of recovery methods to solve local minima problem. Unfortunately these recovery or avoidance methods do not produce optimized path to the goal position, e.g. a techniques developed by Barraquand and Latombe [4]. Their planner *Randomized Path Planner*[4] is following the force specified by the potential function and when stuck at a local minimum it initiated a series of random driving. The meaning of random driving allows randomized path planner to escape the local minimum and in that case, the force is again followed towards the goal position. The drawback of this method is the path produced is not optimal in the sense of shortest path.

6.0.3 Computing the Potential Field

The fundamental building blocks of the potential field is a vector which corresponds to the orientation in which a mobile has to move and also possible to indicate the speed of the robot. As example consider a behavior *SearchGoal* which is a vector that points the robot towards the goal. This collection of vectors is called the potential field because the field causes the robot to be attracted to the goal. It is not necessary to compute the whole field, the robot only compute the artificial force at the its current position, then taking a step in the direction indicated by this force, repeat until goal position is reached or gets stuck in local minimum.

Potential field for two-dimensional navigation is mapping from one vector $v = [x, y]$ into the gradient vector $\Delta = [\Delta x, \Delta y]$. To generate the fields, define Δx, and Δy in terms of v as follows:

- Let (x_{goal}, y_{goal}) denote the position of the goal. Let s denote the spread of the field. Assume the goal is circular, let r denote the radius of the goal. Let $v = [x, y]$ denote the (x_{robot}, y_{robot}) position of the robot.
- Find the distance from the robot to the goal: $|d| = \sqrt{(x_{goal} - x_{robot})^2 + (y_{robot} - y_{goal})^2}$
- Find the angle between the robot and the goal: $\theta = tan^-1(\frac{y_{goal}-y_{robot}}{x_{goal}-x_{robot}})$
- Set Δx and Δy according to the following:

$$if\ d < r \qquad \Delta x = \Delta y = 0$$

$$if\ r \leq d \leq s + r \qquad \Delta x = (d - r)\cos(\theta)\ \ and\ \ \Delta y = (d - r)\sin(\theta)$$

$$if\ d > s + r \qquad \Delta x = s\ \cos(\theta)\ \ and\ \ \Delta y = s\ \sin(\theta)$$

When the robot reaches the goal no forces from the goal affects the robot, which is why both Δx and Δy are set to 0 when $d < r$. The field has a spread and the robot reaches the extent of this field when $d = s + r$. Outside of this

this field the vector size is set to the maximum possible value. Within the field, but outside the goals radius, the vector size is set proportional to the distance between the robot and the goal. Now a method is defined so that the robot would be attracted to the goal, but another behaviour is needed such as the *AvoidObstacles* behaviour. The *AvoidObstacles* potential field is defined as follows:

- Let $(x_{obstacle}, y_{obstacle})$ denote the position of the obstacle. Let r denote the radius of the obstacle. Let $v = [x, y]$ denote the (x, y) position of the robot
- Find the distance between the robot and the obstacle: $|d| = \sqrt{(x_{obstacle} - x_{robot})^2 + (y_{robot} - y_{obstacle})^2}$
- Find the angle between the robot and the obstacle: $\theta = tan^-1(\frac{y_{obstacle} - y_{robot}}{x_{obstacle} - x_{robot}})$
- Set Δx and Δy according to the following:

$$if\ d < r \qquad \Delta x = \ -sign(\cos(\theta))\infty \ \ and \ \ \Delta y = \ -sign(\sin(\theta))\infty$$

$$if\ r \leq d \leq s + r \qquad \Delta x = (s + r - d)\cos(\theta) \ \ and \ \ \Delta y = (s + r - d)\sin(\theta)$$

$$if\ d > s + r \qquad \Delta x = \Delta y = 0$$

Inside an obstacle the repulsive force is infinite and points out from the center of the obstacle. Outside of the obstacles potential field, the repulsive force is zero. Within the potential field but outside the radius of the obstacle the size of the vector corresponds to the distance from the obstacle to the robot. When $s + r - d = 0$ or $s + r = d$ corresponds to the robot being on the edge of the potential field. When $d = r$ corresponding to the robot being on the edge of the obstacle. Notice that the vector points away from the obstacle by introducing the negative sign into definitions of Δx and Δy. Now the potential fields are created there is a need of combining them, this can be done by adding multiple potential fields together. Doing this with all obstacle potential fields and goal potential field gives a new potential field. This field is generated by first finding Δx_{goal} and Δy_{goal}, the vector generated by the attractive goal and finding $\Delta x_{obstacle}$ and $\Delta y_{obstacle}$. The vector generated by the repulsive obstacle and adding these vectors together to find:

$$\Delta x \ = \ \Delta x_{obstacle} \ + \ \Delta x_{goal} \tag{6.1}$$

$$\Delta y \ = \ \Delta y_{obstacle} \ + \ \Delta y_{goal} \tag{6.2}$$

The robot determines the Δx using Equation (6.1) and Δy using Equation(6.2), finds the velocity $v = \sqrt{(\Delta x^2 \ + \ \Delta y^2)}$ and the angle $\theta = tan^-1(\frac{\Delta y}{\Delta x})$, set the speed to v and the direction to θ. as the robot moves in the environment it identifies new force vectors, and chooses new directions and speed. The described approached is based on the following source [12], [34] and [33]

Chapter 7

Path Planning

In the previous chapters three methods for representing the environment has been descried namely visibility graphs section 4.1, cell decomposition chapter 5 and potential functions chapter 6. The two methods visibility graphs 4.1 and cell decomposition 5 approaches are complete which means that it is possible to find a solution if one exists otherwise they report failure. The third method potential functions 6 is a heuristic method that is able to produce a fairly acceptable solution to the problem in many scenarios, but for which there is no formal proof of its correctness.

7.1 Complete Motion Planning

To be able to provide complete motion planning the solution must adopt one of the two following solutions visibility graphs 4.1 or cell decomposition 5 but to be able to provide the Euclidean shortest path the solution can only use visibility graphs 4.1. The visibility graph of a set of non intersecting polygonal obstacles in the plane is an undirected graph whose vertices are the vertices of the obstacles and whose edges are pairs of vertices (u, v) such that the line segment between u and v does not intersect any of the obstacles. Given this graph and two vertices initial and goal position the problem is now reduced to generate a path such that the total path cost of the path is minimal among all possible paths from initial to goal position. Any minimal cost path will do, and the path should consist of a list of connections from the initial vertex to the goal vertex. In many cases edges in graphs are labelled by cost and the cost of a path is defined as the sum of the costs attached to its edge.

Various algorithms have been developed for planning paths. A selection of such algorithms is described in detail in various textbooks such as [26], [20], [11]. Uninformed search methods are whether depth-first or breadth-first, methods for finding paths. In principle, these methods provide a solution to the path-finding problem, but they are often infeasible to use to control path finding systems because the search expands too many vertexes before a path is found. The $A*$ algorithm [26] [20] [10] is a classical and very well known algorithm. $A*$ is guaranteed to return a path of minimum cost wherever a path exists and to return failure otherwise under the specifications given here:

7.2 A* Algorithm

$A*$ explores the graph G iterative by following paths originates from the initial vertex V_{init}. In the beginning of each iteration there are vertices that the algorithm has already visited and there may be other vertices that are still unvisited. For each visited vertex V the previous iterations have produced one or several paths connecting V_{init} to V the algorithm only stores a representation of a path of minimum cost. At any instant the set of all such paths forms a *spanning tree* T of the subset of G so far explored. T is represented by associating to each visited vertex V a pointer to its parent vertex in the current T. $A*$ assigns a cost function $f(V)$ to every vertex in the current T. This function is an estimate of the cost of the minimum cost path in the graph G connecting V_{init} to V_{goal} going through N. The function $f(V)$ is computed as follows:

$$f(V) = g(V) + h(V) \tag{7.1}$$

where:

- $g(V)$ represents the true cost of the path from the initial vertex V_{init} to the current vertex V.
- $h(V)$ represents the estimated cost in moving from vertex V to the goal vertex V_{goal} in G.

If this estimate is exactly equal to the distance along the optimal path the $A*$ algorithm expands to very few vertices the reason is that $A*$ is computing

$f(V) = g(V) + h(V)$ at every vertex. When $h(V)$ exactly matches $g(V)$ the value of $f(V)$ does not change along the path. All vertices that are not on the right path will have a higher value of f than vertices that are on the right path. Since $A*$ does not consider higher valued f vertices until the algorithm has considered lower valued f vertices the $A*$ has a good possibility to not strays off the shortest path.

7.2.1 Choosing an Estimate

The $A*$ algorithm searches a graph efficiently, with respect to the chosen estimate; if the estimate is accurate then the search is efficient. If the estimate is inaccurate although a path will be found its search will take more time than probably required and possibly return a suboptimal path. If we are able to compute a perfect estimate one that always returns the exact minimum path distance between two vertices the algorithm will perform efficient in $O(n)$ where n is the number of steps in the graph. The problem of calculating the exact distance between two vertices is the same as finding the shortest path between the two vertices - which is the problem we are trying to solve in the first place. For non perfect estimates $A*$ behaves slightly differently depending on whether the estimate is too high or too low.

7.2.2 Underestimating

If the estimate is too low so that it underestimates the actual path length the algorithm takes longer to run because $A*$ will prefer to examine vertices closer to the start vertex, rather than vertices close to the goal. This will increase the time it takes the algorithm to find the path through to the goal. If the estimate underestimates in all possible cases then the result that the algorithm produces will be the best path possible. If the estimates ever overestimates this guarantee is lost. In applications where accuracy is more important than performance, it is important to ensure that the estimate is underestimating. In the majority of the literature about path planning in commercial and academic problems, accuracy is often very important and therefore also the approach of underestimating [20] [21] [22] [11].

7.2.3 Overestimating

If the estimate is too high so that it overestimating the actual path $A*$ may not return the optimal path. $A*$ will tend to generate a path with fewer vertices in it, even if the edges between vertices are more costly to travel. The $A*$ will pay less attention to the cost so far travelled and will tend to favor vertices that have less distance to travel. This will move the focus of the search toward the goal faster but include the risk of missing a more efficient path. This means that the total length of the path may be longer than the most efficient path. This does not mean that an overestimated $A*$ algorithm is producing poor paths, it can be shown that if a estimate is overestimated by at most some value x for any vertices in the graph, then the final path will be no more than x too long. This does not mean that the $A*$ cannot be used with an overestimate, but it refers to the fact that the algorithm no longer returns the shortest path.

Most of the literature researched for this thesis aim for estimates that are close, but err on the side of underestimating. The simplest and most common estimate is Euclidean distance [22].

7.2.4 Euclidean Distance

A common estimate is the Euclidean distance which is guaranteed to be underestimating [22]. The cost of an edge between two vertices is given by the distance between the representative vertices. The Euclidean distance is measured directly between two vertices even if there is no direct path because of blocking obstacles. Euclidean distance is always either accurate or an underestimate. Moving around obstacles can only add extra distance. If there is no obstacles between two vertices the estimate is accurate, otherwise it is an underestimate.

7.2.5 Summary

Until now the previous chapters has touched a number of ways to characterize the motion planning problem and the algorithm used to address it. The mobile robot control software environment is already chosen to be Carmen [23] this thesis will therefore not include an analysis and evaluation of various robotics software environments. Instead this thesis will focus on the methods used by Carmen to clarify if Carmen is able to deliver complete path planning and provide the shortest path for a robot.

Chapter 8

Carmen Robot Navigation Toolkit

Carmen [23] is an open source collection of robot control software developed in 2002 by a group of people at Carnegie Mellon University in USA. Its mainly written in the C and C++ programming language, it consists of 640 files and more than 111600 lines of code. It is developed to provide a "consistent interface and a basic set of primitives for robotic research" [23] mainly focusing on single robot control. Carmen supports a large number of different robot platforms like Scout, XR4000, Segway and iRobot just to mention a few of the most famous platforms. It uses a three layered architecture; the first layer is the hardware interface providing low-level control integration of sensor and motion data, the second layer is concerning all basis robot tasks such as motion planning, obstacle avoidance and localization etc. and the third layer is the user-defined application which relies on the functionality of the lower levels. Carmen also provides a configuration tools, a simulator and a map editor tool. Carmen was designed as modular software architecture, with each major capability built as a separate module. Modules communicate with each other over a communication protocol called Inter Process Communication (IPC) [8] which provides message passing between processes. Carmen supports autonomous mobile robot navigation in indoor environments. It contains software modules for collision avoidance, localization, mapping, path planning, navigation and people tracking [23].

8.0.6 Map Files

The essence of the Carmen navigation routines are based on metric environment maps Figure 8.1 depict such a map file. This map file is a real world example taken from a senior care center in Pittsburgh; the map is included in Carmen software package. Map files in Carmen stores an *evidence grid* representation of two dimensional environment as well as other information related to robot navigation and localization. The evidence grid approach represents the robots environment by a two dimensional regular grid. In each cell is stored the evidence or probability based on accumulated sensor readings that describe a particular grid cells accessibility. In modern literature this approach is often referred to as the occupancy grid where each cell holds a probability value that indicates if the cell is either occupied, free or undiscovered. Occupancy grids were first defined by Hans Moravec and Alberto Elfes [24].

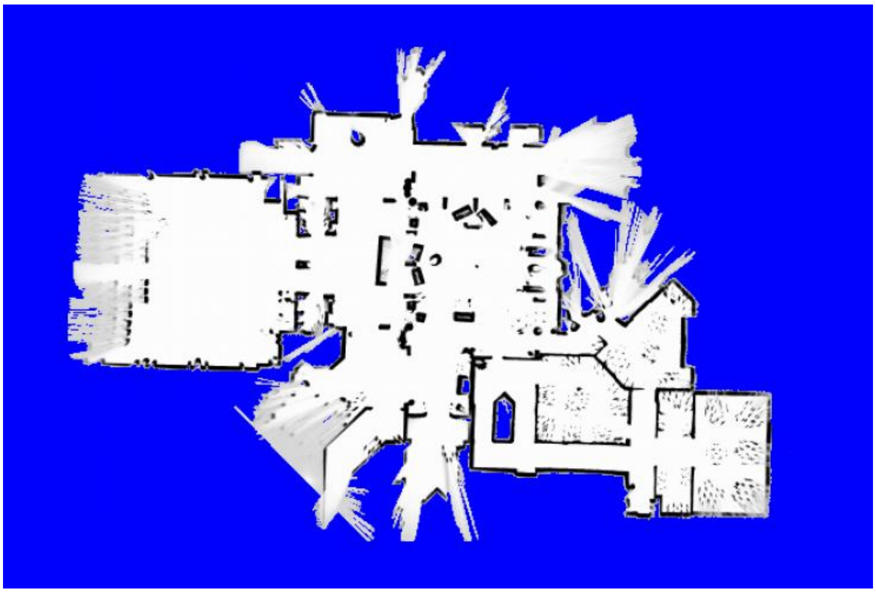

Figure 8.1: Carmen map file of from the Long Wood senior care center

8.1 Path Planning in Carmen

The author Kurt Konolige is making the statement in article [18] that the described method unlike potential field methods in general, have the characteristic that it is impossible to get stuck in a local minimum. Since local minimum other than the goal position are major cause of inefficiency for potential field methods, an important question is the following: Is it possible to construct a potential function which only has one local minimum a goal position. Such a function, if it exists is called a "global navigation function"

> *Computing local-minima-free potentials in large dimensional spaces is a difficult problem that is at least as hard as path planning itself [17].*

In chapter 8 it has been described how to construct a navigation function, i.e. a potential field. To construct the potential field with no local minimum other than the goal configuration is a difficult problem that has a known solution only when the C-obs have simple shapes [20]. The computation of a numerical navigation function over a representation of the C-space in form of a grid turns out to be a easier problem. That is probably the main reasons why Carmen is using a grid approach, since it is difficult to construct an navigation function over a C-free of arbitrary geometry.

According to the article [23] the current path planning implementation in Carmen is an implementation of Konoliges Linear Programming Navigation gradient method or LPN [18]. This section is describing the method presented in [18] the LPN gradient method is based on a numerical artificial potential field as described in chapter 6. The article shows how to build a navigation function that assigns a potential field value to every point in the space. Moving along the gradient of the navigation potential yields the minimum cost path to the goal set. The method samples the C-space and assumes that C-space is to be discretized into a set of points. Values at non-sampled points are being computed by interpolation between sample points.

8.1.1 Path Cost

The purpose is the find a path with minimum cost to some goal point. A path is represented by an ordered set (8.1) of sampling points.

$$P = p_1, p_2, p_3, \; ... \; p_n \tag{8.1}$$

Where the points are contiguous in either rectilinearly or diagonally and no points are repeated and the last point in the set must be the goal point. A cost function $F(P)$ for a path P can be divided into the sum of a cost of being at a point, along with an adjacency cost of moving from one point to the next one.

$$F(P_k) = \sum_i I(p_i) + \sum_i A(p_i, p_{i+1}) \tag{8.2}$$

Where A and I can be arbitrary functions. I will represent the cost of traversing through a given point, and will be set according to how close the robot is too obstacles or have higher value for unknown regions, slippery regions, etc. While A is proportional to the Euclidean distance between two points, then the sum of A gives a cost proportional to the path length.

8.1.2 Navigation Function

A navigation function is the assignment of a potential field value to every point of the C-space, such that the goal point is always attracting the robot everywhere in the environment. Navigation functions unlike potential field methods in general have the characteristic that it is impossible to get stuck in a local minimum [18]. The value of the navigation function N_k in a point p_k is the cost of the minimum cost path that starts from that point.

$$N_k = min_{j=1,\ \dots\ ,m}\ F(P_k^j) \tag{8.3}$$

Where P_k^j is the j-th path starting from point p_k and reaching the goal set point and m is the number of such paths. Calculating the navigation function N_k (8.3) for every point in the C-space directly, would require a very high computational time. Since the size of the C-space can be large, and finding the minimal cost path in a naive manner involves iteration over all paths from the point.

The LPN algorithm [18] is used to efficiently compute the navigation function. It is a generalization of the *wavefront algorithm* [43] [1] the algorithm is based on the following three steps:

- Assign all goal set points a value 0, and every other point an infinite cost
- Put the goal set point in an *active list*
- At each iteration of the algorithm, operate on each point of the active list, removing it from the list and updating its eight neighbours

Consider an arbitrary point p that have assigned a value this point is surrounded by eight neighbours on a regular grid as in Figure 8.2. The process is repeated until the active list is empty. To update a point p are the following operations used:

- For every point q in the eight neighbours of p compute its cost to extend the path from p to this point and the intrinsic cost of q using Equation (8.2)
- If the new value for q is less than the previous one, update the value for q and put it into the active list

The navigation function is computed according to the intrinsic cost of the sampling points in the C-space.

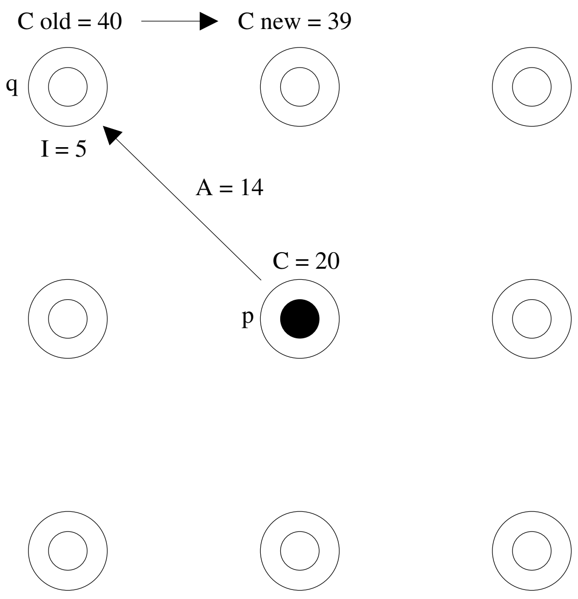

Figure 8.2: Updating the cost of points in the neighbourhood of an active point.

8.1.3 Obstacle Cost

Suppose obstacles in the environment are given by a set of obstacle sampling points. Let Q be a generic function and $d(p)$ the Euclidean distance for the sampling point p from the closest sample point representing an obstacle then:

$$I(p) = Q(d(p)) \tag{8.4}$$

In order to compute $d(p)$ for every sampling point involves the LPN algorithm by giving the obstacle sampling points as the final configuration set, and assigning in the initialization phase a value 0 to the intrinsic cost for all the sampling points. Once $d(p)$ and then $I(p)$ is computed for every sampling point p the LPN algorithm can again be executed to compute the navigation function.

8.2 Evaluating Planning Approach in Carmen

The article [18] referred in this section describing the gradient method a method for local navigation. Potential field methods are often referred to as local methods. This comes from the fact that most potential functions are defined in such a way that their values at any configuration do not depend on the distribution and shapes of the C-obs beyond some limited neighbourhood. If however it could be possible to construct an "ideal" potential function with a single minimum at the goal position. This function could be regarded as some kind of "global" information about the C-free, and the expression "local" would then be less relevant[20].

Most planning methods based on potential field approach like the one used in Carmen are incomplete, they may fail to find a free path, even if one exists.[20]. On the other hand they are particular fast in a wide range of situations. The strength of this approach is that, with some engineering like the "random driving" approaches it makes it possible to construct motion planners which are both quite efficient and reasonably reliable. This explains why they are increasingly popular for implementing practical motion planners.

The potential field planner has successfully solved difficult problems [3] [9]. However it is easy to create problems that it fails to solve in a reasonable amount of time, though they admit rather obvious solutions [2]. Most failures are caused by local minimum the problem is that it is difficult in general to ensure that the potential function will not contain multiple local minima. The robot could become trapped at a local minimum that is not a goal position. In the research for this thesis I identified four significant problems; the problems are based on experimental work with Carmen and problems described in the literature.

- Trap situations due to local minimum
- No passage between closely spaced obstacles
- Oscillations in a narrow passage
- Oscillations in the presence of obstacles

8.3 New Planning Approach in Carmen

The problems encountered with the potential field and thereby with Carmen are leading to develop another planner. The planner in Carmen is a heuristic method, in the new planning approach the requirement from the section 1.2 a complete planner. To develop a new planning approach in Carmen one of the first steps is to analysis the map files to verify is the map files include suitable data to develop a new planner. The map files in Carmen stores a occupancy grid representation of two dimensional environment as well as other information related to robot navigation and localization. The format of these map files is a tagged file format, similar to the TIFF standard for image files. The map file begins with an arbitrary number of lines all starting with ♯, these lines are comments. The documentation of Carmen is very limited though is the official description of the Carmen map file format available on the web page for Carmen [41].

To be able to read the information stored inside the map files, if a new planning approach is needed I need to be able to read the map files. To be able to do that I need to figure out how Carmen itself reads the map files. Carmen is using a library file for reading the maps, it is located at `/carmen/lib/libmap_io.a`. Then apply to the following function `carmen_map_read_gridmap_chunk()` in file `map_io.c`:

```c
int carmen_map_read_gridmap_chunk(char *filename, carmen_map_t
    *map);
```

When the function `carmen_map_read_gridmap_chunk()` is evoked including an appropriate map variable, it is filled with data from the map. The map variable should be a pointer to a `carmen_map_t` structure:

```c
typedef struct {
 carmen_map_config_t config;
 float* complete_map;
 float** map;
} carmen_map_t, *carmen_map_p;
```

The `config` substructure is defined as:

```c
typedef struct {
 int x_size;
 int y_size;
 double resolution;
 char *map_name;
} carmen_map_config_t, *carmen_map_config_p;
```

The map representation is a two dimensional array in $[x], [y]$ it describes the occupancy values at each grid cell. Values between 0.00 and 0.99 are free in the sense of its value. 1.00 corresponds to occupied grid cells and the value -1 is unexplored grid cells and is treated as occupied grid cells. The output from running the `readmap` application:

```
jht@jht:~/01_readmap/readmap_reviewed ./test_map longwood.map
Map in longwood.map is 536 x 379, with resolution 0.10 x 0.10 m/grid cell
The pixel at the centre of the map is 268, 189 and has value 0.27
The total grid-points in longwood.map is: 203144
```

The result of running the `readmap` application on the map file `longwood.map` provides the following information, the resolution is 0.10 x 0.10 m a precise position of the obstacles in the map is not given, only a value 1.00 for a solid obstacle. Since the resolution is 0.10 x 0.10 m we can only be sure to determine the placement of obstacles within a margin error of 0.10 m. This is not precise enough to continue implementing an exact shortest path approach, since the robot in these methods is moving very close to the obstacles in the environment.

The maps and code are based on a software package Carmen, the source files `map_io.c` and `test_map.c` can be found in the appendix.

Chapter 9

Recommended approach for Bispebjerg

When evaluating the techniques described in this thesis it is important to also pay attention to the environment Bispebjerg Hospital where the robot has to operate. This chapter will sum up the advantages and disadvantages of metric and topological approach to mapping in the Table 9.1 and based on the description from section 1.1 this section will try to create an idea of the best suited approach for the environment at Bispebjerg Hospital. A map of the tunnel system can be seen in the Figure 9.1. The tunnel system is a fairly simple structure, if the cargo its not life-threatening material we will properly been satisfied with a suboptimal path and the effort in finding the precise shortest path is less important. In the tunnel environment many hallways and corners may look the same, and with the topological approach it can be hard to distinguish between different places. The robot can easily mistake one position for the other. These two reasons indicate that a metric approach will be the preferred method to this specific hospital environment. A following test drive either in the real environment with a physical robot or a simulation in the map of the tunnel system will indicate the risk of ending up in a local minimum.

Other articles are proponent for an approach that integrates both paradigms: metric and topological, by combining both paradigms the approach presented in this article [44] claims that it gains advantages from both worlds accuracy and

Metric	Topological
+ esay to build, represent and maintain + recognition of places is based on geometry and is non-ambiguous - planning inefficient/ space consuming - requires accurate determination of the robots position	+ permits efficient planning, low space complexity + does not require accurate determination of the robots position - difficulties in constructing and maintaining in large scale environments, if sensor information is ambiguous - sensitive to the point of view, recognition of places is often difficult

Table 9.1: Advantages and disadvantages of metric and topological approach to mapping.

efficiency. Optimality, completeness and computational complexity naturally trade off with each other. Using the method described in the article [44] we must be willing to accept increased computational complexity if we demand optimal and complete planners.

The metric approach has a tendency to be space consuming, another challenge when making use of the metric method is choosing the resolution of the C-space grid. Larger resolution means a smaller space to search and therefore faster methods. On the other hand smaller resolution means a more precise solution and closer to an optimal path. A hierarchical solution may be best choice, where the robot starts with a coarse grid and refines the grid as needed e.g. when the robot is entering a critical section, near crossroads etc. The research of the hierarchical approach is not covered in this thesis.

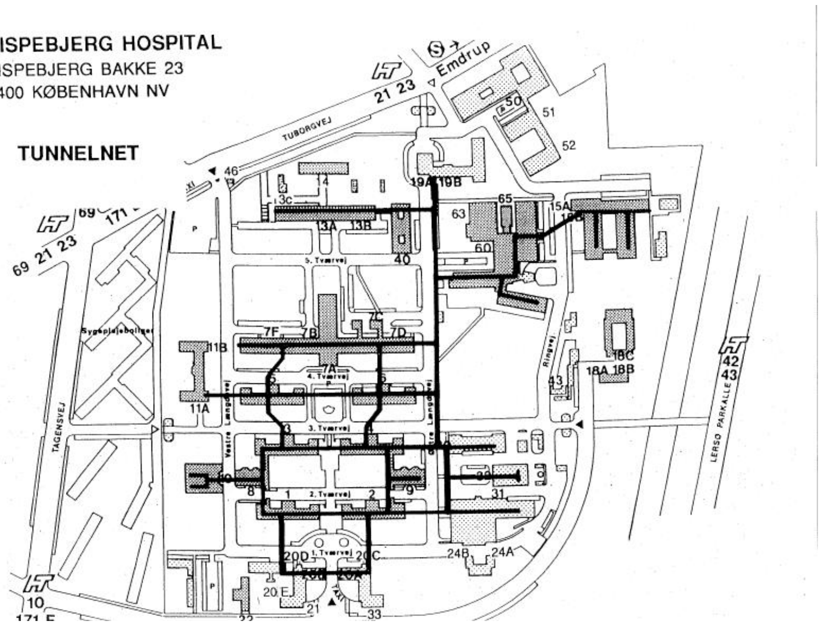

Figure 9.1: The black solid lines illustrates the tunnel system of Bispebjerg Hospital.

Chapter 10

Conclusion

In this thesis I have examined the possibilities of creating a robotic transportation system to solve logistic tasks in a hospital environment. Since approximately 83 percent of hospital budgets are salaries, an automation of logistics tasks will reduce the need of human transportation tasks and release hands for better patient care by the increasing volume of patients with no corresponding increase in staff [32]. The overall goal is to create autonomous robots that will accept high-level descriptions of tasks and will execute them without further human intervention. The input description will specify what the user wants done rather than how to do it. Making progress toward autonomous robots is a practical interest not only in a hospital environment but in a wide variety of application domains includes construction, assistance for the disabled, manufacturing, and space exploration.

It is also of great technical interest, especially for Computer Science because it raises many important problems. One of them is path planning which is the central theme of this thesis. The minimum to expect from an autonomous robot is the ability to plan its own motion. At the first impression of motion planning it can look relatively simple, since people deal with it with no apparent difficulty in their everyday lives; it turns out to be extremely difficult to duplicate this behaviour using a computer controlled robot. The unaware reader will be surprised by the amount of non-trivial mathematical and algorithmic techniques that are necessary to build a reasonably general and reliable path planner.

This thesis introduces the concept of *configuration space* it is used throughout this thesis in order to organize the various aspects of motion planning. The concept is essentially a representational tool, it consists of treating the robot as a point in an appropriate space. Various tools from Geometry, Topology and Algebra apply nicely to this representation and provide the theoretical basic of path planning. Chapter 2 describes the notation of the configuration space for a rigid object that can move and rotate freely among obstacles in a two dimensional environment, the type of robot considered in the basic path planning problem. The chapter also describes how the obstacles contained in the environment are mapped into the configuration space. The concept of configuration space is a tool for formulating path planning problems precisely. In chapter 3 two paradigms for mapping indoor environments are discussed, many factors need to be observed and evaluated to decide what type of approach to use. These include ease of construction, type of environment, storage requirements, equipment availability/cost, processing power requirement and suitability to the task. Metric maps which stores information regarding to the obstacles in the environment are relative easy to construct but need large amount of storage and can be computationally complex due to the number of grid cells. Topological maps are abstract representations of metric maps; topological maps are generated by partitioning the grid-based map into recognizable regions. The topological map is consistent with the grid-based map. For every abstract plan generated using the topological map, there exists a corresponding plan in the grid-based map. Typically, when using abstract models efficiency is traded off with consistency and performance loss.

Chapter 4, 5 and 6 describe three computational approaches for solving the basic path planning problem. Chapter 4 presents a roadmap approach called visibility graph, which consists of capturing the global topology of the set of collision free paths for the robot in form of a graph structure. Chapter 5 describes the cell decomposition approach, which represents the set of collision free configurations as a collection of cells and search the graph representing the adjacency relation among these cells. Chapter 6 describes the potential field approach, where the robot reacts as a particle moving under the action of forces generated by an artificial potential field attracting the robot toward the goal while a repulsing force directing the robot away from obstacles. The roadmap and cell decomposition methods reduce the problem of finding a continuous free path to that of searching a graph.

Chapter 7 introduces path finding with the A$*$ algorithm. A$*$ is a very efficient algorithm and has lots of scope for optimization, many path finding systems are using some variation of the A∗ as its key algorithm [22]. The property of the robot is highly dependent of the choice of the best suited approach, such as wheel settings, turning radius and the dimensions of the robot etc. If the robot is installed with precise hardware for measuring odometric then a metric approach could be the best approach. On the other hand if the money is spend on optimal sensors able to distinguish and detect landmarks in the hospital environment, then the topological approach could be a better approach. Again the best approach depends heavily on the robot and the hardware installed on the robot.

The vast majority of the articles which I have studied during the preparation of this thesis has built a robot which is able to perform a given task like the tour-guide robot RHINO [6]. Based on the choice of hardware installed on the robot, task specification, the environment and basic requirements, after all these issues has been decided or analysed the choice of path planning approach can begin. Developing the technologies required for autonomous robots are a major challenge which involve many important problems, we see that many mobile robot applications are being built from scratch including the RHINO [6], HELPMATE [19] and TUG [14], this makes the automation process relative expensive. Research is needed in this area to find general solutions and to find robust, reliable and safe methods and algorithms.

Carmen is one approach for providing robot motion control but is limited to the potential field method; this technique has a limited ability to deal with local minima of the potential field function. The method is usually incomplete i.e. it may fail to find a free path, even if one exists. The advantage is that in a wide range of situations some of them are particularly fast. After installing Carmen and simply setting up different path planning task for the robot simulator in Carmen, it is obvious that Carmen has problems. The robot has a tendency to get trapped in complex concavities formed by a collection of obstacles; the combination can act as a local minimum. Typically these trapped situations are caused by the fact that the various points which have fixed positions are concurrently attracted toward their goal position. Hence they are competing among themselves to attain their goal positions. To prevent this problem one can design a potential field that has no local minimum other than the goal configuration in the C-free. Another approach is to complement the potential field method with mechanisms for escaping local minimum; this could be one of the following approaches:

- Backtrack when encountering local minimum

- Take a number of random walks when encountering local minimum
- Invoke an alternative planner, such as a wall follower
- Increase the potential of obstacles, so that the robot is repulsed from the local minima

There are benefits of using the potential fields:

- Paths do not need to be planned, but are generated in real time
- Planning and control are merged in one function
- Paths are smooth, the robot does not brush along obstacles.
- Potential field can be dynamically updated
- To speed up computation, only local obstacles can be taken into account

When using the potential field method the effort must be put in developing a method for avoiding local minimum or escape from local minimum. The concept of the precise "shortest path" in terms of distance is given up since the robot is not moving close to the obstacles. A shortest path would not exist in terms of distance, as it will always be possible to shorten a path by moving closer to obstacles. Also the concept of "complete planner" is given up, since the potential field is a local method, at each step they move from one grid configuration to another. It is typical for this method to depend only on the content of the C-space in the neighbourhood of the current configuration of the robot. Therefore potential field methods are often called local methods, while the roadmap and cell decomposition methods are called global methods. We must be willing to accept increased computation complexity if we demand optimal motion plans or completeness from our planner. One could argue that "shortest path" and "complete planner" are not important in the robotic project at Bispebjerg Hospital; if that is the case then the potential field method could be the best approach.

There are a number of interesting path planning concepts not covered in detail in this thesis, such as navigation among moving obstacles and coordination of multiple robots.

Appendix A

map_io.c

```
/******************************************************
 *
 * This source code is part of the Carnegie Mellon Robot
 * Navigation Toolkit (CARMEN)
 *
 * CARMEN Copyright (c) 2002 Michael Montemerlo, Nicholas
 * Roy, and Sebastian Thrun
 *
 * CARMEN is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public
 * License as published by the Free Software Foundation;
 * either version 2 of the License, or (at your option)
 * any later version.
 *
 * CARMEN is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied
 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
 * PURPOSE.  See the GNU General Public License for more
 * details.
 *
 * You should have received a copy of the GNU General
 * Public License along with CARMEN; if not, write to the
 * Free Software Foundation, Inc., 59 Temple Place,
 * Suite 330, Boston, MA  02111-1307 USA
 *
 ******************************************************/

#include "map_io.h"
```

```c
#include <assert.h>
#include <stdarg.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <inttypes.h>

#if __BYTE_ORDER == __BIG_ENDIAN
#warning
#warning This processor is big-endian, and you will
#warning not be able to read maps correctly.
#warning Please use a different computer.
#warning
#error
#endif

void
carmen_warn(char* fmt, ...) {
  va_list args;

  va_start(args, fmt);
  vfprintf(stderr, fmt, args);
  va_end(args);
  fflush(stderr);
}

#define carmen_test_alloc(X) do {if ((void *)(X) == NULL)
    carmen_die("Out of memory in %s, (%s, line %d).\n",
    __FUNCTION__, __FILE__, __LINE__); } while (0)

void
carmen_die(char* fmt, ...) {
  va_list args;

  va_start(args, fmt);
  vfprintf(stderr, fmt, args);
  va_end(args);
  fflush(stderr);
  exit(-1);
}

int
carmen_file_exists(char *filename)
{
  FILE *fp;

  fp = fopen(filename, "r");
  if(fp == NULL)
    return 0;
  else {
    fclose(fp);
    return 1;
  }
}

/*
```

```
 * if size < 0 then ignore buf and advance fp to one byte after the next '\0'
 */
static int read_string(char *buf, int size, carmen_map_file_p fp)
{
  int i;
  char c;

  for (i = 0; i < size || size < 0; i++) {
    c = carmen_map_fgetc(fp);
    if (c == EOF)
      return -1;
    if (c == '\0') {
      if (size >= 0)
  buf[i] = '\0';
      return i;
    }
    if (size >= 0)
      buf[i] = c;
  }

  buf[size-1] = '\0';
  return -1;
}

int carmen_map_read_comment_chunk(carmen_map_file_p fp)
{
  char comment_str[100], c;
  char *err, id[1024];

  do {
    c = carmen_map_fgetc(fp);
    if(c == EOF)
      return -1;
    else if(c == '#')
      {
  err = carmen_map_fgets(comment_str, 100, fp);
  if(err == NULL)
    return -1;
      }
    else
      carmen_map_fseek(fp, -1, SEEK_CUR);
  } while(c == '#');

  assert (strlen(CARMEN_MAP_LABEL)+strlen(CARMEN_MAP_VERSION) < 1024);

  if(carmen_map_fread(&id, strlen(CARMEN_MAP_LABEL)+
          strlen(CARMEN_MAP_VERSION), 1, fp) == 0)
    return -1;
  if(strncmp(id, CARMEN_MAP_LABEL, strlen(CARMEN_MAP_LABEL)) != 0)
    return 1;
  if(strncmp(id+strlen(CARMEN_MAP_LABEL), CARMEN_MAP_VERSION,
       strlen(CARMEN_MAP_VERSION)) != 0)
    return 1;
```

```c

  return 0;
}

int carmen_map_file(char *filename)
{
  carmen_map_file_p fp;
  int comment_return;

  if(!carmen_file_exists(filename))
    return 0;
  fp = carmen_map_fopen(filename, "r");
  if (fp == NULL)
    return 0;

  comment_return = carmen_map_read_comment_chunk(fp);
  carmen_map_fclose(fp);

  if (!comment_return)
    return 1;

  return 0;
}

int carmen_map_name_chunk(char *in_file, char *out_file, int
    chunk_type, char *chunk_name)
{
  unsigned char *buf;
  int size, type, retval;
  carmen_map_file_p in_fp, out_fp;

  in_fp = carmen_map_fopen(in_file, "r");

  if (in_fp == NULL) {
    carmen_warn("Error: Can't open file %s for reading", in_file);
    return -1;
  }

  if (carmen_map_advance_to_chunk(in_fp, chunk_type) < 0) {
    carmen_warn("Error: Can't advance to chunk %d", chunk_type);
    carmen_map_fclose(in_fp);
    return -1;
  }

  type = carmen_map_fgetc(in_fp);
  if(type == EOF) {
    carmen_warn("Error: Unexpected EOF");
    carmen_map_fclose(in_fp);
    return -1;
  }

  if (CARMEN_MAP_CHUNK_IS_NAMED(type)) {
    carmen_warn("Error: Chunk is already named");
    carmen_map_fclose(in_fp);
    return -1;
  }

  if (carmen_map_fread(&size, sizeof(int), 1, in_fp) < 1) {
    carmen_map_fclose(in_fp);
    return -1;
  }

  buf = (unsigned char *) calloc(size, sizeof(char));
  carmen_test_alloc(buf);

  carmen_map_fread(buf, 1, size, in_fp);

  carmen_map_fclose(in_fp);

  retval = carmen_map_named_chunk_exists(out_file, chunk_type,
      chunk_name);
  if (retval < 0)
    return -1;
  if (retval > 0) {
    carmen_warn("Error: Map chunk of type %d named %s already
        exists in file %s", chunk_type, chunk_name, out_file);
    return -1;
  }

  out_fp = carmen_map_fopen(out_file, "a");
  if (out_fp == NULL) {
    carmen_warn("Error: Can't open file %s for appending",
        out_file);
    return -1;
  }

  carmen_map_fputc(chunk_type | CARMEN_MAP_NAMED_CHUNK_FLAG,
      out_fp);
  size += strlen(chunk_name) + 1;
  carmen_map_fwrite(&size, sizeof(int), 1, out_fp);
  carmen_map_fwrite(buf, sizeof(char), 10, out_fp);
  carmen_map_fprintf(out_fp, "%s", chunk_name);
  carmen_map_fputc('\0', out_fp);
  size -= 10 + strlen(chunk_name) + 1;
  carmen_map_fwrite(buf+10, sizeof(char), size, out_fp);

  carmen_map_fclose(out_fp);
  return 0;
}

void strip_trailing_spaces(char *str, int len)
{
  int i;

  i = len - 1;
  while(i >= 0 && str[i] == ' ')
    i--;
  i++;
  if(i == len)
    i--;
  str[i] = '\0';
}

int carmen_map_advance_to_chunk(carmen_map_file_p fp, int
    specific_chunk)
{
  int chunk_type, chunk_size, done = 0;

  carmen_map_fseek(fp, 0, SEEK_SET);
  if(carmen_map_read_comment_chunk(fp) < 0) {
    fprintf(stderr, "Error: Could not read comment chunk.\n");
    return -1;
  }

  specific_chunk &= ~(unsigned char)CARMEN_MAP_NAMED_CHUNK_FLAG;

  do {
    chunk_type = carmen_map_fgetc(fp);
    if(chunk_type == EOF)
      done = 1;
    chunk_type &= ~(unsigned char)CARMEN_MAP_NAMED_CHUNK_FLAG;
    if(chunk_type == specific_chunk) {
      carmen_map_fseek(fp, -1, SEEK_CUR);
      return 0;
    }
    if(!done)
      if(carmen_map_fread(&chunk_size, sizeof(int), 1, fp) < 1)
  done = 1;
    if(!done)
      if(carmen_map_fseek(fp, chunk_size, SEEK_CUR) < 0)
  done = 1;
    if(!done && carmen_map_feof(fp))
      done = 1;
  } while(!done);
  return -1;
}

int carmen_map_advance_to_named_chunk(carmen_map_file_p fp, int
    specific_chunk, char *name)
{
  int chunk_type, named, chunk_size, len, done = 0;
  char buf[128];

  carmen_map_fseek(fp, 0, SEEK_SET);
  if(carmen_map_read_comment_chunk(fp) < 0) {
    fprintf(stderr, "Error: Could not read comment chunk.\n");
    return -1;
  }

  specific_chunk &= ~CARMEN_MAP_NAMED_CHUNK_FLAG;

  do {
    chunk_type = carmen_map_fgetc(fp);
    if(chunk_type == EOF)
      done = 1;
    named = CARMEN_MAP_CHUNK_IS_NAMED(chunk_type);
    chunk_type &= ~CARMEN_MAP_NAMED_CHUNK_FLAG;
    if(chunk_type == specific_chunk && named) {
      carmen_map_fseek(fp, 14, SEEK_CUR);
      len = read_string(buf, 128, fp);
      if (len < 0)
  return -1;
      if (!strcmp(buf, name)) {
  carmen_map_fseek(fp, -15-len-1 , SEEK_CUR);
  return 0;
      }
      else
  carmen_map_fseek(fp, -14-len-1 , SEEK_CUR);
    }
    if(!done)
      if(carmen_map_fread(&chunk_size, sizeof(int), 1, fp) < 1)
  done = 1;
    if(!done)
      if(carmen_map_fseek(fp, chunk_size, SEEK_CUR) < 0)
  done = 1;
    if(!done && carmen_map_feof(fp))
      done = 1;
  } while(!done);
  return -1;
}

int carmen_map_chunk_exists(char *filename, int specific_chunk)
{
  carmen_map_file_p fp;
  char chunk_type;
  int chunk_size;

  fp = carmen_map_fopen(filename, "r");
  if(fp == NULL) {
    fprintf(stderr, "Error: could not open file %s for reading.\n",
      filename);
    return -1;
  }
  if(carmen_map_advance_to_chunk(fp, specific_chunk) < 0) {
    carmen_map_fclose(fp);
    return 0;
  }
  else {
    chunk_type = carmen_map_fgetc(fp);
    carmen_map_fread(&chunk_size, sizeof(int), 1, fp);
    carmen_map_fclose(fp);
    return chunk_size;
  }
}

int carmen_map_named_chunk_exists(char *filename, int
    specific_chunk, char *name)
{
  carmen_map_file_p fp;
  char chunk_type;
```

```c
  int chunk_size;

  fp = carmen_map_fopen(filename, "r");
  if(fp == NULL) {
    fprintf(stderr, "Error: could not open file %s for reading.\n",
      filename);
    return -1;
  }
  if(carmen_map_advance_to_named_chunk(fp, specific_chunk, name) <
      0) {
    carmen_map_fclose(fp);
    return 0;
  }
  else {
    chunk_type = carmen_map_fgetc(fp);
    carmen_map_fread(&chunk_size, sizeof(int), 1, fp);
    carmen_map_fclose(fp);
    return chunk_size;
  }
}

static int carmen_map_read_creator_chunk_data(carmen_map_file_p
    fp, time_t *creation_time,
                char *username, char *origin,
                char *description)
{
  carmen_map_fread(username, 10, 1, fp);
  strip_trailing_spaces(username, 10);
  carmen_map_fread(creation_time, sizeof(time_t), 1, fp);
  carmen_map_fread(origin, 80, 1, fp);
  strip_trailing_spaces(origin, 80);
  carmen_map_fread(description, 80, 1, fp);
  strip_trailing_spaces(description, 80);

  carmen_map_fclose(fp);
  return 0;
}

int carmen_map_read_creator_chunk(char *filename, time_t
    *creation_time,
          char *username, char *origin,
          char *description)
{
  carmen_map_file_p fp;
  int chunk_type, chunk_size;
  char chunk_description[12];

  fp = carmen_map_fopen(filename, "r");
  if(fp == NULL) {
    fprintf(stderr, "Error: could not open file %s for reading.\n",
      filename);
    return -1;
  }
  if(carmen_map_advance_to_chunk(fp, CARMEN_MAP_CREATOR_CHUNK) < 0)
    {
      if (carmen_map_advance_to_chunk(fp, 0) < 0)
  {
    carmen_warn("You have an old-style map. The creator chunk id "
          "is wrong.\n\nFIX IT!\n\n");
  }
      else {
  carmen_warn("Error: Could not find a creator chunk.\n");
  carmen_map_fclose(fp);
  return -1;
      }
    }
  chunk_type = carmen_map_fgetc(fp);
  carmen_map_fread(&chunk_size, sizeof(int), 1, fp);
  carmen_map_fread(chunk_description, 10, 1, fp);

  if (CARMEN_MAP_CHUNK_IS_NAMED(chunk_type)) {
    if (read_string(NULL, -1, fp) < 0) {
      carmen_warn("Error: Unexpected EOF.\n");
      carmen_map_fclose(fp);
      return -1;
    }
  }

  return carmen_map_read_creator_chunk_data(fp, creation_time,
      username, origin, description);
}

int carmen_map_read_named_creator_chunk(char *filename, char
    *chunk_name, time_t *creation_time,
          char *username, char *origin,
          char *description)
{
  carmen_map_file_p fp;
  int chunk_type, chunk_size;
  char chunk_description[12];

  fp = carmen_map_fopen(filename, "r");
  if(fp == NULL) {
    fprintf(stderr, "Error: could not open file %s for reading.\n",
      filename);
    return -1;
  }
  if(carmen_map_advance_to_named_chunk(fp,
      CARMEN_MAP_CREATOR_CHUNK, chunk_name) < 0)
    {
      if (carmen_map_advance_to_chunk(fp, 0) < 0)
  {
    carmen_warn("You have an old-style map. The creator chunk id "
          "is wrong.\n\nFIX IT!\n\n");
  }
      else {
  carmen_warn("Error: Could not find a creator chunk named
      \"%s\"\n", chunk_name);
  carmen_map_fclose(fp);
  return -1;
```

```
      }
    }
  chunk_type = carmen_map_fgetc(fp);
  carmen_map_fread(&chunk_size, sizeof(int), 1, fp);
  carmen_map_fread(chunk_description, 10, 1, fp);

  if (CARMEN_MAP_CHUNK_IS_NAMED(chunk_type)) {
    if (read_string(NULL, -1, fp) < 0) {
      carmen_warn("Error: Unexpected EOF.\n");
      carmen_map_fclose(fp);
      return -1;
    }
  }

  return carmen_map_read_creator_chunk_data(fp, creation_time,
      username, origin, description);
}

static int carmen_map_read_gridmap_config_data(carmen_map_file_p
    fp, carmen_map_config_p config)
{
  int size_x, size_y;
  float resolution;

  carmen_map_fread(&size_x, sizeof(int), 1, fp);
  carmen_map_fread(&size_y, sizeof(int), 1, fp);
  carmen_map_fread(&resolution, sizeof(float), 1, fp);

  config->x_size = size_x;
  config->y_size = size_y;
  config->resolution = resolution;

  carmen_map_fclose(fp);
  return 0;
}

int carmen_map_read_gridmap_config(char *filename,
    carmen_map_config_p config)
{
  carmen_map_file_p fp;
  int chunk_type, chunk_size;
  char chunk_description[12];

  if (config == NULL)
    {
      fprintf(stderr, "Error: config argument is NULL in %s.\n",
        __FUNCTION__);
      return -1;
    }

  fp = carmen_map_fopen(filename, "r");
  if(fp == NULL) {
    fprintf(stderr, "Error: could not open file %s for reading.\n",
      filename);
    return -1;
  }

  if(carmen_map_advance_to_chunk(fp, CARMEN_MAP_GRIDMAP_CHUNK) < 0) {
    fprintf(stderr, "Error: Could not find a gridmap chunk.\n");
    fprintf(stderr, "       This file is probably not a map file.\n");
    carmen_map_fclose(fp);
    return -1;
  }
  chunk_type = carmen_map_fgetc(fp);
  carmen_map_fread(&chunk_size, sizeof(int), 1, fp);
  carmen_map_fread(chunk_description, 10, 1, fp);

  if (CARMEN_MAP_CHUNK_IS_NAMED(chunk_type)) {
    if (read_string(NULL, -1, fp) < 0) {
      carmen_warn("Error: Unexpected EOF.\n");
      carmen_map_fclose(fp);
      return -1;
    }
  }

  config->map_name = (char *)calloc(strlen(filename)+1, sizeof(char));
  carmen_test_alloc(config->map_name);
  strcpy(config->map_name, filename);

  return carmen_map_read_gridmap_config_data(fp, config);
}

int carmen_map_read_named_gridmap_config(char *filename, char *chunk_name, carmen_map_config_p config)
{
  carmen_map_file_p fp;
  int chunk_type, chunk_size;
  char chunk_description[12];

  if (config == NULL)
    {
      fprintf(stderr, "Error: config argument is NULL in %s.\n",
        __FUNCTION__);
      return -1;
    }

  fp = carmen_map_fopen(filename, "r");
  if(fp == NULL) {
    fprintf(stderr, "Error: could not open file %s for reading.\n",
      filename);
    return -1;
  }

  if(carmen_map_advance_to_named_chunk(fp, CARMEN_MAP_GRIDMAP_CHUNK, chunk_name) < 0) {
    fprintf(stderr, "Error: Could not find a gridmap chunk named \"%s\"\n", chunk_name);
    fprintf(stderr, "        This file is probably not a map
        file.\n");
    carmen_map_fclose(fp);
    return -1;
  }
  chunk_type = carmen_map_fgetc(fp);
  carmen_map_fread(&chunk_size, sizeof(int), 1, fp);
  carmen_map_fread(chunk_description, 10, 1, fp);

  if (CARMEN_MAP_CHUNK_IS_NAMED(chunk_type)) {
    if (read_string(NULL, -1, fp) < 0) {
      carmen_warn("Error: Unexpected EOF.\n");
      carmen_map_fclose(fp);
      return -1;
    }
  }

  config->map_name = (char *)calloc(strlen(chunk_name)+1,
      sizeof(char));
  carmen_test_alloc(config->map_name);
  strcpy(config->map_name, chunk_name);

  return carmen_map_read_gridmap_config_data(fp, config);
}

static int carmen_map_read_gridmap_chunk_data(carmen_map_file_p
    fp, carmen_map_p map)
{
  int size_x, size_y;
  float resolution;
  int n;

  carmen_map_fread(&size_x, sizeof(int), 1, fp);
  carmen_map_fread(&size_y, sizeof(int), 1, fp);
  carmen_map_fread(&resolution, sizeof(float), 1, fp);

  map->config.x_size = size_x;
  map->config.y_size = size_y;
  map->config.resolution = resolution;

  map->complete_map =
    (float *)calloc(map->config.x_size * map->config.y_size,
        sizeof(float));
  carmen_test_alloc(map->complete_map);
  map->map = (float **)calloc(map->config.x_size, sizeof(float *));
  carmen_test_alloc(map->map);
  for(n = 0; n < map->config.x_size; n++)
    map->map[n] = map->complete_map + n * map->config.y_size;
  carmen_map_fread(map->complete_map, sizeof(float) * size_x *
      size_y, 1, fp);

  carmen_map_fclose(fp);
  return 0;
}

```

```c
/*
 * Call "carmen_map_read_gridmap_chunk" function with an appropriate
 * map variable, it is filled in with map data. The map variable
 * should be a pointer to a carmen_map_t structure
 */

int carmen_map_read_gridmap_chunk(char *filename, carmen_map_p map)
{
  carmen_map_file_p fp;
  int chunk_type, chunk_size;
  char chunk_description[12];

  fp = carmen_map_fopen(filename, "r");
  if(fp == NULL) {
    fprintf(stderr, "Error: could not open file %s for reading.\n",
            filename);
    return -1;
  }
  if(carmen_map_advance_to_chunk(fp, CARMEN_MAP_GRIDMAP_CHUNK) < 0) {
    fprintf(stderr, "Error: Could not find a gridmap chunk.\n");
    fprintf(stderr, "       This file is probably not a map file.\n");
    carmen_map_fclose(fp);
    return -1;
  }
  chunk_type = carmen_map_fgetc(fp);
  carmen_map_fread(&chunk_size, sizeof(int), 1, fp);
  carmen_map_fread(chunk_description, 10, 1, fp);

  chunk_description[10] = '\0';

  if (CARMEN_MAP_CHUNK_IS_NAMED(chunk_type)) {
    if (read_string(NULL, -1, fp) < 0) {
      carmen_warn("Error: Unexpected EOF.\n");
      carmen_map_fclose(fp);
      return -1;
    }
  }

  map->config.map_name = (char *)calloc(strlen(filename)+1, sizeof(char));
  carmen_test_alloc(map->config.map_name);
  strcpy(map->config.map_name, filename);

  return carmen_map_read_gridmap_chunk_data(fp, map);
}

int carmen_map_read_named_gridmap_chunk(char *filename, char *chunk_name, carmen_map_p map)
{
  carmen_map_file_p fp;
  int chunk_type, chunk_size;
  char chunk_description[12];

  fp = carmen_map_fopen(filename, "r");
  if(fp == NULL) {
    fprintf(stderr, "Error: could not open file %s for reading.\n",
            filename);
    return -1;
  }
  if(carmen_map_advance_to_named_chunk(fp,
      CARMEN_MAP_GRIDMAP_CHUNK, chunk_name) < 0) {
    fprintf(stderr, "Error: Could not find a gridmap chunk named
        \"%s\"\n", chunk_name);
    carmen_map_fclose(fp);
    return -1;
  }
  chunk_type = carmen_map_fgetc(fp);
  carmen_map_fread(&chunk_size, sizeof(int), 1, fp);
  carmen_map_fread(chunk_description, 10, 1, fp);

  if (CARMEN_MAP_CHUNK_IS_NAMED(chunk_type)) {
    if (read_string(NULL, -1, fp) < 0) {
      carmen_warn("Error: Unexpected EOF.\n");
      carmen_map_fclose(fp);
      return -1;
    }
  }

  map->config.map_name = (char *)calloc(strlen(chunk_name)+1,
      sizeof(char));
  carmen_test_alloc(map->config.map_name);
  strcpy(map->config.map_name, chunk_name);

  return carmen_map_read_gridmap_chunk_data(fp, map);
}

static int carmen_map_read_places_chunk_data(carmen_map_file_p fp,
    carmen_map_placelist_p places)
{
  int i;
  int place_type;
  float float_var;

  carmen_map_fread(&(places->num_places), sizeof(int), 1, fp);
  places->places = (carmen_place_p)calloc(places->num_places,
              sizeof(carmen_place_t));
  carmen_test_alloc(places->places);
  for(i = 0; i < places->num_places; i++) {
    carmen_map_fread(&place_type, sizeof(int), 1, fp);
    carmen_map_fread(places->places[i].name, 20, 1, fp);
    strip_trailing_spaces(places->places[i].name, 20);
    if(place_type == CARMEN_NAMED_POSITION_TYPE) {
      carmen_map_fread(&float_var, sizeof(float), 1, fp);
      places->places[i].x = float_var;
      carmen_map_fread(&float_var, sizeof(float), 1, fp);
      places->places[i].y = float_var;
    }
    else if(place_type == CARMEN_NAMED_POSE_TYPE) {
      carmen_map_fread(&float_var, sizeof(float), 1, fp);
      places->places[i].x = float_var;
      carmen_map_fread(&float_var, sizeof(float), 1, fp);
      places->places[i].y = float_var;
      carmen_map_fread(&float_var, sizeof(float), 1, fp);
      places->places[i].theta = float_var;
    }
    else if(place_type == CARMEN_LOCALIZATION_INIT_TYPE) {
      carmen_map_fread(&float_var, sizeof(float), 1, fp);
      places->places[i].x = float_var;
      carmen_map_fread(&float_var, sizeof(float), 1, fp);
      places->places[i].y = float_var;
      carmen_map_fread(&float_var, sizeof(float), 1, fp);
      places->places[i].theta = float_var;

      carmen_map_fread(&float_var, sizeof(float), 1, fp);
      places->places[i].x_std = float_var;
      carmen_map_fread(&float_var, sizeof(float), 1, fp);
      places->places[i].y_std = float_var;
      carmen_map_fread(&float_var, sizeof(float), 1, fp);
      places->places[i].theta_std = float_var;
    }
    places->places[i].type = place_type;
  }

  carmen_map_fclose(fp);
  return 0;
}

int carmen_map_read_places_chunk(char *filename,
    carmen_map_placelist_p places)
{
  carmen_map_file_p fp;
  int chunk_type, chunk_size;
  char chunk_description[12];

  fp = carmen_map_fopen(filename, "r");
  if(fp == NULL) {
    fprintf(stderr, "Error: could not open file %s for reading.\n",
      filename);
    return -1;
  }
  if(carmen_map_advance_to_chunk(fp, CARMEN_MAP_PLACES_CHUNK) < 0)
      {
    fprintf(stderr, "Error: Could not find a places chunk.\n");
    carmen_map_fclose(fp);
    return -1;
  }
  chunk_type = carmen_map_fgetc(fp);
  carmen_map_fread(&chunk_size, sizeof(int), 1, fp);
  carmen_map_fread(chunk_description, 10, 1, fp);

  if (CARMEN_MAP_CHUNK_IS_NAMED(chunk_type)) {
    if (read_string(NULL, -1, fp) < 0) {
      carmen_warn("Error: Unexpected EOF.\n");
      carmen_map_fclose(fp);
      return -1;
    }
  }

  return carmen_map_read_places_chunk_data(fp, places);
}

int carmen_map_read_named_places_chunk(char *filename, char
    *chunk_name,
               carmen_map_placelist_p places)
{
  carmen_map_file_p fp;
  int chunk_type, chunk_size;
  char chunk_description[12];

  fp = carmen_map_fopen(filename, "r");
  if(fp == NULL) {
    fprintf(stderr, "Error: could not open file %s for reading.\n",
      filename);
    return -1;
  }
  if(carmen_map_advance_to_named_chunk(fp,
      CARMEN_MAP_PLACES_CHUNK, chunk_name) < 0) {
    fprintf(stderr, "Error: Could not find a places chunk named
        \"%s\"\n", chunk_name);
    carmen_map_fclose(fp);
    return -1;
  }
  chunk_type = carmen_map_fgetc(fp);
  carmen_map_fread(&chunk_size, sizeof(int), 1, fp);
  carmen_map_fread(chunk_description, 10, 1, fp);

  if (CARMEN_MAP_CHUNK_IS_NAMED(chunk_type)) {
    if (read_string(NULL, -1, fp) < 0) {
      carmen_warn("Error: Unexpected EOF.\n");
      carmen_map_fclose(fp);
      return -1;
    }
  }

  return carmen_map_read_places_chunk_data(fp, places);
}

static int carmen_map_read_laserscans_chunk_data(carmen_map_file_p
    fp,
             carmen_laser_scan_p *scan_list,
             int *num_scans)
{
  int i, j;
  float float_var;

  carmen_map_fread(num_scans, sizeof(int), 1, fp);
  *scan_list =
    (carmen_laser_scan_p)calloc(*num_scans,
        sizeof(carmen_laser_scan_t));
  carmen_test_alloc(*scan_list);
  for(i = 0; i < *num_scans; i++) {
    carmen_map_fread(&float_var, sizeof(float), 1, fp);
    (*scan_list)[i].x = float_var;
    carmen_map_fread(&float_var, sizeof(float), 1, fp);
    (*scan_list)[i].y = float_var;
    carmen_map_fread(&float_var, sizeof(float), 1, fp);
    (*scan_list)[i].theta = float_var;

    carmen_map_fread(&((*scan_list)[i].num_readings), sizeof(int),
        1, fp);

    (*scan_list)[i].range = (float
        *)calloc((*scan_list)[i].num_readings,
              sizeof(float));
    carmen_test_alloc((*scan_list)[i].range);
    for (j = 0; j < (*scan_list)[i].num_readings; j++) {
      carmen_map_fread(&float_var, sizeof(float), 1, fp);
      (*scan_list)[i].range[j] = float_var;
    }
  }

  carmen_map_fclose(fp);
  return 0;
}

int carmen_map_read_laserscans_chunk(char *filename,
           carmen_laser_scan_p *scan_list,
           int *num_scans)
{
  carmen_map_file_p fp;
  int chunk_type, chunk_size;
  char chunk_description[12];

  fp = carmen_map_fopen(filename, "r");
  if(fp == NULL) {
    fprintf(stderr, "Error: could not open file %s for reading.\n",
      filename);
    return -1;
  }
  if(carmen_map_advance_to_chunk(fp, CARMEN_MAP_LASERSCANS_CHUNK)
      < 0) {
    fprintf(stderr, "Error: Could not find a laserscans chunk.\n");
    carmen_map_fclose(fp);
    return -1;
  }
  chunk_type = carmen_map_fgetc(fp);
  carmen_map_fread(&chunk_size, sizeof(int), 1, fp);
  carmen_map_fread(chunk_description, 10, 1, fp);

  if (CARMEN_MAP_CHUNK_IS_NAMED(chunk_type)) {
    if (read_string(NULL, -1, fp) < 0) {
      carmen_warn("Error: Unexpected EOF.\n");
```

```c
      carmen_map_fclose(fp);
      return -1;
    }
  }

  return carmen_map_read_laserscans_chunk_data(fp, scan_list, num_scans);
}

int carmen_map_read_named_laserscans_chunk(char *filename, char *chunk_name,
               carmen_laser_scan_p *scan_list,
               int *num_scans)
{
  carmen_map_file_p fp;
  int chunk_type, chunk_size;
  char chunk_description[12];

  fp = carmen_map_fopen(filename, "r");
  if(fp == NULL) {
    fprintf(stderr, "Error: could not open file %s for reading.\n",
      filename);
    return -1;
  }
  if(carmen_map_advance_to_named_chunk(fp, CARMEN_MAP_LASERSCANS_CHUNK, chunk_name) < 0) {
    fprintf(stderr, "Error: Could not find a laserscans chunk named \"%s\"\n", chunk_name);
    carmen_map_fclose(fp);
    return -1;
  }
  chunk_type = carmen_map_fgetc(fp);
  carmen_map_fread(&chunk_size, sizeof(int), 1, fp);
  carmen_map_fread(chunk_description, 10, 1, fp);

  if (CARMEN_MAP_CHUNK_IS_NAMED(chunk_type)) {
    if (read_string(NULL, -1, fp) < 0) {
      carmen_warn("Error: Unexpected EOF.\n");
      carmen_map_fclose(fp);
      return -1;
    }
  }

  return carmen_map_read_laserscans_chunk_data(fp, scan_list, num_scans);
}

static int carmen_map_read_hmap_chunk_data(carmen_map_file_p fp, carmen_hmap_p hmap)
{
  int i, j, k, n = 0;
  float x, y, theta;
  char buf[128];

  carmen_map_fread(&hmap->num_zones, sizeof(int), 1, fp);
  hmap->zone_names = (char **) calloc(hmap->num_zones, sizeof(char *));
  carmen_test_alloc(hmap->zone_names);
  for (i = 0; i < hmap->num_zones; i++) {
    n = read_string(buf, 128, fp);
    if (n < 0) {
      free(hmap->zone_names);
      hmap->zone_names = NULL;
      return -1;
    }
    hmap->zone_names[i] = (char *) calloc(n+1, sizeof(char));
    carmen_test_alloc(hmap->zone_names[i]);
    strncpy(hmap->zone_names[i], buf, n+1);
  }

  carmen_map_fread(&hmap->num_inodes, sizeof(int), 1, fp);
  hmap->inodes = (carmen_hmap_inode_p) calloc(hmap->num_inodes, sizeof(carmen_hmap_inode_t));
  carmen_test_alloc(hmap->inodes);
  for (i = 0; i < hmap->num_inodes; i++) {
    carmen_map_fread(&hmap->inodes[i].type, sizeof(int), 1, fp);
    carmen_map_fread(&hmap->inodes[i].degree, sizeof(int), 1, fp);
    hmap->inodes[i].keys = (int *) calloc(hmap->inodes[i].degree, sizeof(int));
    carmen_test_alloc(hmap->inodes[i].keys);
    carmen_map_fread(hmap->inodes[i].keys, sizeof(int), hmap->inodes[i].degree, fp);
    switch (hmap->inodes[i].type) {
    case CARMEN_HMAP_INODE_DOOR:        n = 2; break;
    case CARMEN_HMAP_INODE_ELEVATOR:    n = 1; break;
    }
    hmap->inodes[i].num_points = n * hmap->inodes[i].degree;
    hmap->inodes[i].points = (carmen_point_p) calloc(hmap->inodes[i].num_points, sizeof(carmen_point_t));
    carmen_test_alloc(hmap->inodes[i].points);
    for (j = 0; j < hmap->inodes[i].degree; j++) {
      for (k = 0; k < n; k++) {
  carmen_map_fread(&x, sizeof(float), 1, fp);
  hmap->inodes[i].points[j*n+k].x = (double) x;
  carmen_map_fread(&y, sizeof(float), 1, fp);
  hmap->inodes[i].points[j*n+k].y = (double) y;
  carmen_map_fread(&theta, sizeof(float), 1, fp);
  hmap->inodes[i].points[j*n+k].theta = (double) theta;
      }
    }
  }

  return 0;
}

int carmen_map_read_hmap_chunk(char *filename, carmen_hmap_p hmap)
{
  carmen_map_file_p fp;
  int chunk_type, chunk_size;
  char chunk_description[12];

  fp = carmen_map_fopen(filename, "r");
  if(fp == NULL) {
    fprintf(stderr, "Error: could not open file %s for reading.\n",
      filename);
    return -1;
  }
  if(carmen_map_advance_to_chunk(fp, CARMEN_MAP_HMAP_CHUNK) < 0) {
    carmen_map_fclose(fp);
    return -1;
  }
  chunk_type = carmen_map_fgetc(fp);
  carmen_map_fread(&chunk_size, sizeof(int), 1, fp);
  carmen_map_fread(chunk_description, 10, 1, fp);

  if (CARMEN_MAP_CHUNK_IS_NAMED(chunk_type)) {
    if (read_string(NULL, -1, fp) < 0) {
      carmen_warn("Error: Unexpected EOF.\n");
      carmen_map_fclose(fp);
      return -1;
    }
  }

  return carmen_map_read_hmap_chunk_data(fp, hmap);
}
```

Appendix B

test_map.c

```c
#include "map_io.h"

int main(int argc, char *argv[])
{
  carmen_map_t map;

  if (argc != 2) {
    fprintf(stderr, "Usage: %s <mapfile>\n", argv[0]);
    exit(-1);
  }

  if (carmen_map_read_gridmap_chunk(argv[1], &map) < 0) {
    fprintf(stderr, "Could not read %s\n", argv[1]);
    exit(-1);
  }

  printf("Map in %s is %d x %d, with resolution %.2f x %.2f m/grid
      cell\n",
   argv[1], map.config.x_size, map.config.y_size,
   map.config.resolution, map.config.resolution);

  printf("The pixel at the centre of the map is %d, %d and has
      value %.2f\n",
   map.config.x_size/2, map.config.y_size/2,
   map.map[map.config.x_size/2][map.config.y_size/2]);

  printf("The total grid-points in %s is: %d\n",
   argv[1], map.config.x_size * map.config.y_size);
```

```

    return 0;
}
```

Bibliography

[1] Ronald C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. *In Robotics and Autonomous Systems*, 6(-), 1990.

[2] Jerome Barraquand, Lydia Kavraki, Jean-Claude Latombe, Rajeev Motwani, Tsai-Yen Li, and Prabhakar Raghavan. A random sampling scheme for path planning. *The International Journal of Robotics Research*, 16(6), 1997.

[3] Jerome Barraquand and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6), 1991.

[4] Jérôme Barraquand, Lydia Kavraki, Rajeev Motwani, Jean-Claude Latombe, Prabhakar Raghavan, and Tsai-Yen Li. A random sampling scheme for path planning. *International Journal of Robotics Research*, 16(6), 1997.

[5] J. D. Boissonnat and M. Yvinec. *Algorithmic Geometry.* Cambridge University Press, 1998.

[6] Joachim Buhmann, Wolfram Burgard, Armin B. Cremers, Dieter Fox, Thomas Hofmann, Frank E. Schneider, Jiannis Strikos, and Sebastian Thrun. The mobile robot rhino. *AI Magazine*, 16(1), 1995.

[7] Wolfram Burgard, Armin B. Cremers, Dieter Fox, Dirk Hähnel, Gerhard Lakemeyer, Dirk Schulz, Walter Steiner, and Sebastian Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 14, 1999.

[8] CARMEN-Team. Inter-process communication (ipc) system. Online: 2009-08-12.

[9] Hsuan Chang and Tsai Yen Li. Assembly maintainability study with motion planning. *Int. Conf. on Robotics and Automation*, 1995.

[10] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion Theory, Algorithms, and Implementations.* The MIT Press, 2004.

[11] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry Algorithms and Applications.* Springer, 2000.

[12] Michael A. Goodrich. Potential fields tutorial.

[13] http://mathworld.wolfram.com/Shear.html, 2009. Online: 2009-12-25.

[14] http://www.aethon.com/default.php, 2009. Online: 2009-04-27.

[15] http://www.iai.uni-bonn.de/ rhino/, 2009. Online: 2009-11-07.

[16] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1), 1995.

[17] D.E. Koditschek. Exact robot navigation by means of potential functions: Some topological considerations. In *Proceedings of IEEE Int. Conf. on Robotics and Automation, Pages 1-6.* IEEE Computer Society, 1987.

[18] Kurt Konolige. A gradient method for realtime robot control. *Intelligent Robots and Systems*, 1(-), 2000.

[19] Bala Krishnamurthy and John Evans. Helpmate: A robotic courier for hospital use, 1992.

[20] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[21] Steven M. LaValle. *Planning Algorithms.* Cambridge University Press, 2006.

[22] Ian Millington. *Artificial Intelligence for Games.* Morgan Kaufmann Publishers, an imprint of Elsevier, 2006.

[23] Michael Montemerlo, Nicholas Roy, and Sebastian Thrun. Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit. In -. Proceedings. IEEE/RSJ International Conference on Intelligent Robot and Systems, 2003.

[24] Hans Moravec, Alberto Elfes, Chuck Thorpe, L. Matthies, R.S. Wallace, Patrick Muir, Dong Hun Shin, and Gregg Podnar. Autonomous mobile robots. In *tech. report CMU-RI-TR-86-04*. Robotics Institute, Carnegie Mellon University, February, 1985.

[25] Issa A.D. Nesnas, Anne Wright, Max Bajracharya, Reid Simmons, and Tara Estlin. Claraty and challenges of developing interoperable robotic software, 2003.

[26] Nils J. Nilsson. *Principles of Artificial Intelligence.* Springer-Verlag, 1982.

[27] Illah Nourbakhsh, Judith Bobenage, Sebastien Grange, Ron Lutz, Roland Meyer, and Alvaro Soto. An affective mobile educator with a full-time job. *Artificial Intelligence*, 14, 1999.

[28] Anders Oreback and Henrik I. Christensen. Evaluation of architectures for mobile robotics. In *Autonomous Robots.* Springer Netherlands, 2003.

[29] Ali Gurcan Ozkil, Steen Dawids, Zhun Fan, and Torben Soerensen. Design of a robotic automation system for transportation of goods in hospitals, 2007.

[30] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[31] John H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science, Pages 421-427.* IEEE Computer Society, 1979.

[32] John Restrepo-Giraldo, Jorgen P. Bansler, Peter Jacobsen, and Henning Boje Andersen. Styring, kvalitet og design i sundhedssektoren, 2009.

[33] Maria Isabel Ribeiro. Obstacle avoidance.

[34] Hani Safadi. Local path planning using virtual potential field, 2007. Online: 2010-01-12.

[35] José Santos-Victor, Raquel Vassallo, and Hans Schneebeli. Topological maps for visual navigation. *Computer Vision Systems*, 1999.

[36] Christian Schlegel. A component approach for robotics software: Communication patterns in the orocos context. In *Fachtagung Autonome Mobile Systeme.* Springer, Karlsruhe, 2003.

[37] Jacob T. Schwartz and Micha Sharir. On the piano movers' problem ii. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 1983.

[38] Jacob T. Schwartz, Micha Sharir, and J.E. Hopcroft. *Planning Geometry and Complexity of Robot Motion (Ablex Series in Artificial Intelligence, Vol 4).* Ablex Publishing Corporation, 1987.

[39] Roland Siegwart, Kai O. Arras, Samir Bouabdallah, Daniel Burnier, Gilles Froidevaux, Xavier Greppin, Björn Jensen, Antoine Lorotte, Laetitia Mayor, Mathieu Meisser, Roland Philippsen, Ralph Piguet, Guy Ramel, Gregoire Terrien, and Nicola Tomatis. Robox at expo.02: A large-scale installation of personal robots. *Journal of Robotics and Autonomous Systems*, 42, 2003.

[40] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots.* Kluwer Academic Publishers, 2004.

[41] Reid Simmons. The map file format of carmen. Online: 2009-08-12.

[42] Michael Sipser. *Introduction to the Theory of Computation.* Cengage Learning, 2005.

[43] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Henning, T. Hofmann, M. Krell, and T. Schmidt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R.P. Bonasso, and R Murphy, editors, *AI-based Mobile Robots: Case Studies of Successful Robot Systems.* MIT Press, 1998.

[44] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 1997.

[45] Richard T. Vaughan, Brian P. Gerkey, and Andrew Howard. On device abstractions for portable, reusable robot code. In -. Proceedings. IEEE/RSJ International Conference on Intelligent Robot and Systems, 2003.

[46] Panagiotis G. Zavlangas and Spyros G. Tzafestas. Integration of topological and metric maps for indoor mobile robot path planning and navigation. *Methods and Applications of Artificial Intelligence*, 2002.