Motion Planning for Mobile Robots - A Guide

S.A.M. Coenen

CST 2012.108

Master's thesis

Coach(es): dr.ir. M.J.G. van de Molengraft
ir. J.J.M. Lunenburg
dr.ir. G.J.L. Naus

Supervisor: prof.dr.ir. M. Steinbuch

Eindhoven University of Technology
Department of Mechanical Engineering
Control Systems Technology

Eindhoven, November, 2012

Contents

List of Notations

The following notations are used throughout this literature survey. The far most of them are introduced in Chapter 2. For some notations the reference to the accompanying equation is included. Most of the time calligraphic letters, e.g., $\mathcal{S}$, denote a set.

$\mathcal{A}$	The robot is called $\mathcal{A}$. Multiple robots are denoted as $\mathcal{A}_i$.
$\mathcal{W}$	The robots' workspace is denoted by $\mathcal{W}$ and is modeled as a Euclidean space $\mathbb{R}^d$, with the dimension $d = 2$ or $d = 3$. $\mathbb{R}$ is the set of real numbers.
$\mathcal{WO}_i$	Obstacles in $\mathcal{W}$ are referred to as $\mathcal{WO}_i$. A particular obstacle is denoted with its index $i = 1, 2, \ldots$.
$\mathcal{WO}$	The union of all obstacles $\mathcal{WO}_i$ is the obstacle region and is denoted as $\mathcal{WO}$.
$\mathcal{C}$	The configuration space of $\mathcal{A}$ is referred to as $\mathcal{C}$. An element of the set $\mathcal{C}$ (i.e., a configuration) is denoted by q. The subset of $\mathcal{W}$ occupied by $\mathcal{A}$ at q is denoted as $\mathcal{A}(q)$.
$\mathcal{CO}_i$	Obstacles in $\mathcal{C}$ are denoted as $\mathcal{CO}_i$. A particular obstacle is denoted with its index $i = 1, 2, \ldots$, see Equation 2.1.
$\mathcal{CO}$	The union of $\mathcal{CO}_i$ is the configuration space obstacle region and is denoted as $\mathcal{CO}$, see Equation 2.2.
$\mathcal{C}_{\text{free}}$	The free configuration space, $\mathcal{C}_{\text{free}}$, is the complement of $\mathcal{CO}$, see Equation 2.3.
$cl(\mathcal{C}_{\text{free}})$	The closure of $\mathcal{C}_{\text{free}}$ is referred to as $cl(\mathcal{C}_{\text{free}})$ and consists of $\mathcal{C}_{\text{free}}$ and the configurations at which the robot contacts $\mathcal{CO}$, denoted as $\mathcal{C}_{\text{contact}}$, see Equation 2.5.
c	A path is denoted as c, see Equation 2.4. It is defined as a function of a parameter s that usually takes a value in $[0, 1]$. If $c \in \mathcal{C}_{\text{free}}$ it is a free path and if $c \in cl(\mathcal{C}_{\text{free}})$ it is a semi-free path. When c is time-dependent it is called a trajectory and denoted as $c(t)$.
$\mathcal{R}$	A roadmap is a network of curves that are in $\mathcal{C}_{\text{free}}$ and is defined as $\mathcal{R}$.
$G(N, E)$	A search graph is denoted as G and consist of a set of nodes N and a set of edges E that connect nodes.
$\mathcal{CT}$	The configuration space extended with a time dimension is called the configuration-time space and is denoted as $\mathcal{CT}$.
$\mathcal{S}$	A state s encodes the robot's configuration and velocity. The space of all states, or state space, is denoted as $\mathcal{S}$.
$\mathcal{ST}$	The state space extended with a time dimension is called the state $\times$ time space and is denoted as $\mathcal{ST}$.

Chapter 1

Introduction

Moving from one place to another is a trivial task, for humans. One decides how to move in a split second. For a robot such an elementary and basic task is a major challenge. In autonomous robotics motion planning is one of the most significant challenges. There is a fundamental need to specify a task in a high-level language, that is automatically translated into low-level descriptions of how the robot should move. The typical problem is to find a motion for a robot, whether it is a vacuum cleaning robot, a robotic arm, or a magically flying object, from a starting position to a goal position whilst safely avoiding any obstacles in its way.

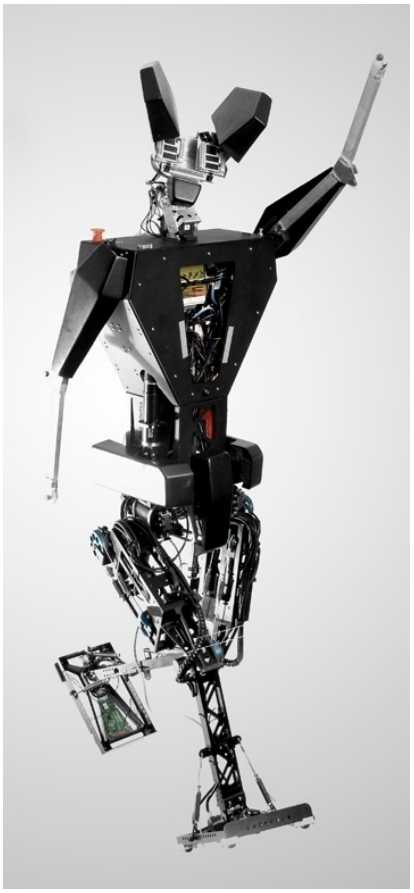
1.1 The RoboCup Project

Eindhoven University of Technology participates in the RoboCup project (Kitano et al., 1997; RoboCup, 2012), an international research and education initiative, that fosters artificial intelligence and robotics research. RoboCup provides a standard problem where a wide range of technologies can be integrated and researched, as well as being used for integrated project-oriented education. For this purpose, at its start in 1997 RoboCup chose to use the soccer game as its primary research platform. The RoboCup Soccer League was formed. Its goal concerns cooperative, fully autonomous multi-robot and multi-agent systems in dynamic, adversarial environments. Its aim is to

> *"develop a team of fully autonomous robots that can win against the human world champion team in soccer by 2050"*(RoboCup, 2012).

In 2000 also the @Home League and the Search & Rescue League were set up. The goal of the RoboCup @Home League is to develop autonomous service and assisting robot technology with high relevance for domestic applications. The intention of the RoboCup Rescue League is to promote research and development in the disaster rescue domain. Eindhoven University of Technology competes in the RoboCup project under the flag of team Tech United Eindhoven (Tech United Eindhoven, 2012) against other universities over the world in the Soccer League and @Home League. Tech United Eindhoven participates in the RoboCup Soccer Middle Size League (MSL) with its TURTLE (Tech United RoboCup Team Limited Edition) soccer robots, depicted in Figure 1.1a and in the RoboCup Soccer Humanoid League with its humanoid soccer robot TUlip, depicted in Figure 1.1b. In the @Home League Tech United Eindhoven participates with its service robot AMIGO (Autonomous Mate for IntelliGent Operations), depicted in Figure 1.1c.

(a) The TURTLE soccer robot.



(b) The TUlip humanoid soccer robot.

(c) The AMIGO service robot.

Figure 1.1: Team Tech United of Eindhoven University of Technology competes in the RoboCup Soccer Middle Size League with its Tech United RoboCup Team Limited Edition (TURTLE) robot and in the RoboCup Soccer Humanoid League with its humanoid soccer robot TUlip. In the RoboCup @Home League it participates with its Autonomous Mate for IntelliGent Operations (AMIGO) service robot.

1.2 Problem Description

The goal of this work is two-fold. First of all it attempts to give a summary of the motion planning problem and the most recent techniques to solve it for an autonomous, mobile robot. Secondly, it provides a framework for the selection of an appropriate motion planner, given the problem and the robot. This framework consist of the formulation of a set of requirements and the comparison of the available motion planners with respect to those requirements. To summarize, the following subgoals are defined:

- ▷ Perform a thorough literature study to order the vast amount of available motion planners.
- ▷ Formulate requirements to obtain a basis for the selection of an appropriate motion planner, given the problem and a robot.
- ▷ Compare the discussed motion planners with respect to the requirements.

The robots of Tech United that compete in the RoboCup are examples of autonomous, mobile robots. They serve as a demonstrator that will show how to select a new motion planner given a problem and a set of requirements. The TURTLE robot and AMIGO currently use a motion planner. For now there is no necessity for a motion planner for the TUlip robot as its development concerns stable walking and kicking of the ball. Although the currently used motion planners for the TURTLE robot and AMIGO have been implemented, tested and used successfully (de Best et al., 2010; Dirkx, 2011), the selection of these planners is mainly based on best practices and common knowledge of motion planning algorithms. For both robots, various points for improvement are identified, which are difficult or even impossible to solve using the current motion planners. A new motion planner must be selected and implemented that solves the current problems. Hereto, the following subgoals are identified:

- ▷ Formulate the requirements for the TURTLE robot and AMIGO.
- ▷ Propose a new motion planner implementation based on the requirements and the comparison of motion planners.
- ▷ Implement this new motion planner and compare it to the existing implementation.

1.3 Outline

This study starts with the introduction of the motion planning problem in Chapter 2. Besides the problem statement, the general framework to tackle the problem is presented. To solve the motion planning problem a search must be conducted in a complex world. Therefore a representation of this world is necessary. Given a certain representation, the motion planning problem transforms to the problem of searching this represented world.

In Chapter 3 requirements are formulated to form a basis for the selection of an appropriate motion planner. These requirements will be a guideline in the search for a motion planner that solves a particular motion planning problem. In Chapter 4 the different classes of representations will be discussed and compared based on the requirements. The same is done in Chapter 5 for algorithms that can search the represented world.

Motion planning problems are not just solved by choosing a representation method and a search algorithm. The combination and use of both makes a motion planner that solves a motion planing problem. This part is called the planning approach and it is discussed in Chapter 6.

Chapter 7 will introduce the motion planning problem for both the TURTLEs and AMIGO. The properties of the robots and their environment will be addressed. Furthermore, the current motion planners are introduced. Next, their shortcomings will be shown. Based upon the shortcomings and requirements a new implementation for both robots in proposed.

Chapter 2

The Motion Planning Problem

This chapter elaborates the problem statement. Also, the general framework to tackle the problem is presented.

Finding a motion from a starting position to a goal position whilst safely avoiding any obstacles is referred to as a *motion planning problem*. A widespread view is that this motion planning problem merely consists of some sort of collision checking or obstacle avoidance. In fact, motion planning encompasses a whole lot more than just that. It involves the planning of a collision-free path in an environment that can be (partly) unknown with moving obstacles that have arbitrary geometries and for a robot that can have a complex geometry and has dynamics of its own. Hence, the motion planning problem has to deal with temporal, geometrical and physical constraints.

Algorithms that solve a motion planning problem, from here on referred to as *motion planners*, are part of the navigation system of a robot. A navigation system translates a specified high-level *task* into low-level descriptions of how the robot should move, i.e., a *motion*. In order to plan these low-level descriptions a robot needs a representation of the environment, i.e., a *map*. This map is constructed through *perception* of the environment using sensors. Obstacles must be mapped into this map and the robot needs to localize itself in this map. Summarizing, a robot needs to accomplish three tasks to navigate:

- ▷ localization
- ▷ mapping
- ▷ motion planning

The system in a robot that accomplishes these tasks and provides directions to a destination is called the *navigation system*. This general and simplified version of the architecture of a navigation system is depicted in Figure 2.1. The focus of this survey is solely on motion planning. From here on it is assumed that when a robot navigation system is described it has a localization and mapping module.

A motion planner can solve the motion planning problem in multiple ways. How the problem is solved is explained with an unconstrained and simplified version of the motion planning problem: the *basic motion planning problem* as introduced by Latombe (1990). In Section 2.1 this basic problem is defined. To solve the motion planning problem a search must be conducted in a complex world. The representation of this world is discussed in Section 2.2. Given a certain representation, the motion planning problem transforms to the problem of searching this represented world. Search algorithms are introduced in Section 2.3. The motion planning problem is commonly divided into a global and local planning problem. This distinction is defined in Section 2.4. Finally, as the basic problem is oversimplified and therefore limiting the practicality of the solutions to the problem, in Section 2.5, extensions of the basic problem are treated.

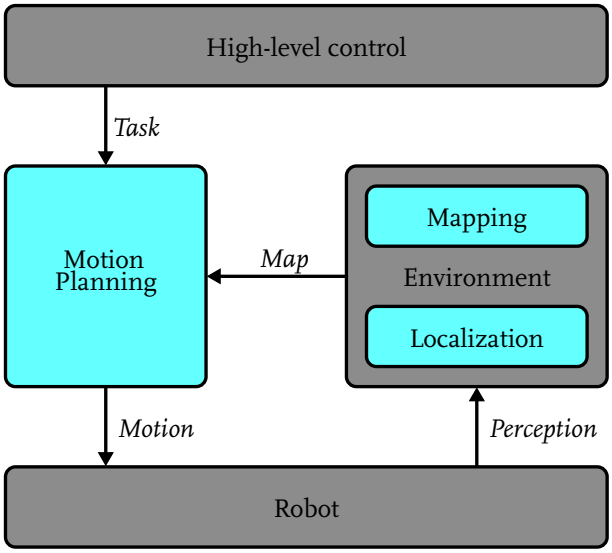


Figure 2.1: Simplified representation of a navigation system of a robot. The three tasks the robot must accomplish (highlighted in cyan) are in between a high-level layer and a low-level layer.

2.1 The Basic Motion Planning Problem

The general motion planning problem can be relaxed to form a basic problem. A static environment is assumed in which a single, rigid body is the only moving object. The dynamic properties of this body are not accounted for and no position or velocity constraints are involved. In a more formal, mathematical sense this basic problem can be defined as a single, rigid body $\mathcal{A}$ that moves in a n-dimensional Euclidean space represented as $\mathcal{W} = \mathbb{R}^d$, with $d = 2$ or $d = 3$, called the *workspace*. Let $\mathcal{O}_i$ for $i = 1, \ldots, n_o$ be a number n_o of static, rigid obstacles in $\mathcal{W}$. These are referred to as $\mathcal{W}$-obstacles or $\mathcal{WO}_i$. The union of all obstacles is called the *obstacle region* and is denoted as $\mathcal{WO}$. Both $\mathcal{A}$ and $\mathcal{WO}_i$ are subsets of $\mathcal{W}$. It is assumed that both the geometry and position of $\mathcal{A}$ and $\mathcal{O}_i$ is known. The problem can now be defined as:

> *Given an initial position and orientation and a goal position and orientation of $\mathcal{A}$ in $\mathcal{W}$, find a path c in the form of a continuous sequence of positions and orientations of $\mathcal{A}$ that do not collide or contact with $\mathcal{O}_i$, that will allow $\mathcal{A}$ to move from its starting position and orientation to its goal position and orientation and report failure if such a path does not exist.*

This problem is known as the *basic motion planning problem* (Latombe, 1990). For a single body moving in $\mathbb{R}^2$ this is also referred to as the piano movers' problem, as it captures the difficulties faced by movers when maneuvering a piano (without lifting it) among obstacles. For a single body moving in $\mathbb{R}^3$, so if the piano is magically free-flying, it is known as the generalized movers' problem (Schwartz and Sharir, 1983). The basic motion planning problem is also referred to as the *path planning problem*, as the assumptions basically transform the physical motion problem into a purely geometric problem. Basic motion planning has evolved through the years to address more complex problems. This evolution allows for the application of motion planning in many different fields such as gaming and entertainment, transportation, autonomous navigation, planetary exploration, demining, industrial production lines, surgery and biological molecular structure analysis (LaValle, 2006). Many solutions of the basic motion planning problem have a straightforward extension into more advanced motion planning problems. Some mobile navigation problems can even be realistically represented in the form of a basic motion planning problem.

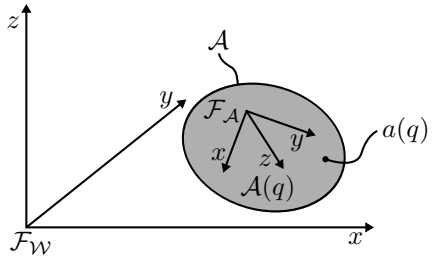


Figure 2.2: A robot $\mathcal{A}$ moves in a $\mathcal{W} = \mathbb{R}^d$, with $d = 2$ or $d = 3$. A configuration of $\mathcal{A}$ specifies the position and orientation of the body-fixed frame $\mathcal{F}_\mathcal{A}$ with respect to workspace frame $\mathcal{F}_\mathcal{W}$. A configuration q of $\mathcal{A}$ is denoted as $\mathcal{A}(q)$. In similar fashion, a point a in $\mathcal{A}(q)$ in $\mathcal{W}$ is denoted as $a(q)$.

2.2 Representing the World

To solve the motion planning problem a search must be conducted in the workspace. Thereto, first the position of the robot is to be specified in an appropriate space. More specifically, every point on the robot must be specified in the space in order to ensure that no point on the robot collides with an obstacle. Hereto, the *configuration space* (Lozano-Pérez, 1983) is introduced. The underlying idea is to represent the robot's configuration as a single point and to map the obstacles in this space. The problem of planning the motion of an arbitrarily shaped robot is transformed to the problem of planning the motion of a point. This level of abstraction allows more explicit constraints on the robot motion. Furthermore, the uniform framework allows for a large range of different motion problems in terms of geometry and kinematics to be tackled by the same planning algorithms.

2.2.1 Concept of the Configuration Space

Consider a single, rigid body $\mathcal{A}$ moving in $\mathcal{W}$, represented as a Euclidean space $\mathbb{R}^d$, with $d = 2$ or $d = 3$, as illustrated in Figure 2.2. $\mathcal{W}$ has a fixed Cartesian coordinate frame, $\mathcal{F}_\mathcal{W}$. $\mathcal{A}$ is represented at a reference position and orientation as a subset of $\mathbb{R}^d$. A body-fixed frame $\mathcal{F}_\mathcal{A}$ is attached to $\mathcal{A}$. A *configuration*, denoted as q, of $\mathcal{A}$ is a specification of the position and orientation of $\mathcal{F}_\mathcal{A}$ with respect to $\mathcal{F}_\mathcal{W}$. The configuration space, denoted as $\mathcal{C}$, is the space of all configurations of the robot. A configuration is simply a point in this abstract configuration space. The subset of the workspace $\mathcal{W}$ that is occupied by a configuration q of $\mathcal{A}$ is denoted as $\mathcal{A}(q)$. In similar fashion, a point a in $\mathcal{A}(q)$ is denoted as $a(q)$. The coordinates that describe a configuration q are generally of two types. Cartesian coordinates are used to describe the position of a body, while angular coordinates are used to represent the rotation of that body. Cartesian coordinates take a value in the Euclidean space $\mathbb{R}$. Angular coordinates take a value in the Special Orthogonal Group $SO(m)$, where $m = 2$ for a planar rotation and $m = 3$ for a spatial rotation. The configuration space of a robot is then obtained in general as a Cartesian product of these two spaces. For example:

- ▷ If the robot is a single point translating in $\mathcal{W} = \mathbb{R}^2$, $\mathcal{C}$ is a plane, and a configuration can be represented using two parameters (x, y).
- ▷ If the robot is a 2-dimensional shape that can translate and rotate, still $\mathcal{W} = \mathbb{R}^2$. However, $\mathcal{C} = \mathbb{R}^2 \times SO(2)$, and a configuration can be represented using three parameters (x, y, θ).
- ▷ If the robot is a 3-dimensional shape that can translate and rotate, $\mathcal{W} = \mathbb{R}^3$ and $\mathcal{C} = \mathbb{R}^3 \times SO(3)$, and a configuration requires six parameters: (x, y, z) for translation, and, e.g., three Euler angles (ϕ, θ, ψ) for rotation.
- ▷ If the robot is a fixed-base manipulator with n revolute joints, $\mathcal{C}$ is n-dimensional.

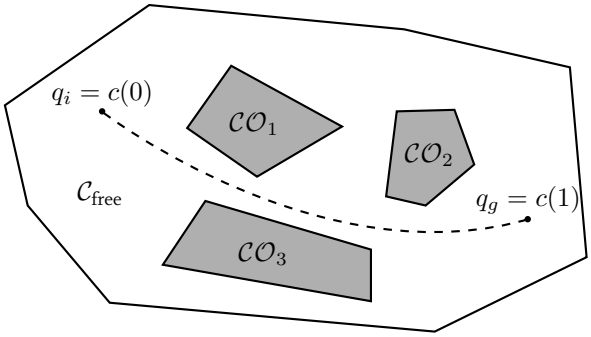


Figure 2.3: A free path connecting q_i to q_g by a curve on the free configuration space $\mathcal{C}_{\text{free}}$.

2.2.2 Obstacles in the Configuration Space

With $\mathcal{C}$ defined, the task is to find a path c in the form of a continuous sequence of configurations of $\mathcal{A}$, from an initial configuration q_i to a goal configuration q_g, that do not collide or contact with $\mathcal{O}_i$. Hereto, the space of configurations for which a collision or contact occurs is defined, by mapping the obstacles in the configuration space. A $\mathcal{W}$-obstacle in $\mathcal{C}$ is called a $\mathcal{C}$-obstacle and is defined as

$$\mathcal{CO}_i = \{q \in \mathcal{C} | \mathcal{A}(q) \cap \mathcal{O}_i \neq \emptyset\}. \tag{2.1}$$

The union of these configuration space obstacles,

$$\mathcal{CO} = \bigcup_{i=1}^{n_o} \mathcal{CO}_i, \tag{2.2}$$

is called the *configuration space obstacle region*. Its complement is

$$\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{CO} \tag{2.3}$$

and is called the *free configuration space*. The basic motion planning problem can now be defined as finding a path from q_i to q_g in $\mathcal{C}_{\text{free}}$. A *path* is defined as a continuous function c that maps a path parameter s (usually taken in unit interval $[0, 1]$) to a curve in $\mathcal{C}$. So a path is defined as continuous function

$$c : [0, 1] \to \mathcal{C} \quad \text{where} \quad c(0) = q_i,\ c(1) = q_g \quad \text{and} \quad c(s) \in \mathcal{C} \quad \forall s \in [0, 1]. \tag{2.4}$$

Analogously a *free path* is defined as a continuous function $c : [0, 1] \to \mathcal{C}_{\text{free}}$, as illustrated in Figure 2.3. That is, if $c(0)$ and $c(1)$ belong to the same connected component of $\mathcal{C}_{\text{free}}$. With $\mathcal{C}_{\text{free}}$ defined as the complement of $\mathcal{CO}$, configurations that belong to both spaces are excluded. The space that contains configurations that represent the robot touching an obstacle is called the contact space and is denoted as $\mathcal{C}_{\text{contact}}$. As this might be allowed or could even be desired these configurations must be represented in $\mathcal{C}_{\text{free}}$ as well for some problems. This space is referred to as the closure of $\mathcal{C}_{\text{free}}$ or $cl(\mathcal{C}_{\text{free}}) = \mathcal{C}_{\text{free}} \cup \mathcal{C}_{\text{contact}}$. Indeed it holds that

$$\mathcal{C}_{\text{free}} \subset cl(\mathcal{C}_{\text{free}}) \subset \mathcal{C}. \tag{2.5}$$

Analogously to the definition of a free path, a continuous function $c : [0, 1] \to cl(\mathcal{C}_{\text{free}})$ is defined as a *semi-free path*.

Obstacles can be modeled in $\mathcal{W} = \mathbb{R}^2$ as convex polygonal regions and in $\mathcal{W} = \mathbb{R}^3$ as convex polyhedral regions. A polygon or polyhedral consists of a finite sequence of straight line segments. In some motion planning problems obstacles are better modeled as *generalized polygons*, i.e., regions bounded by straight segments and/or circular arcs. For the sake of simplicity mostly convex (generalized) polygons in $\mathcal{W} = \mathbb{R}^2$ are treated. Obstacles shapes can also be approximated.

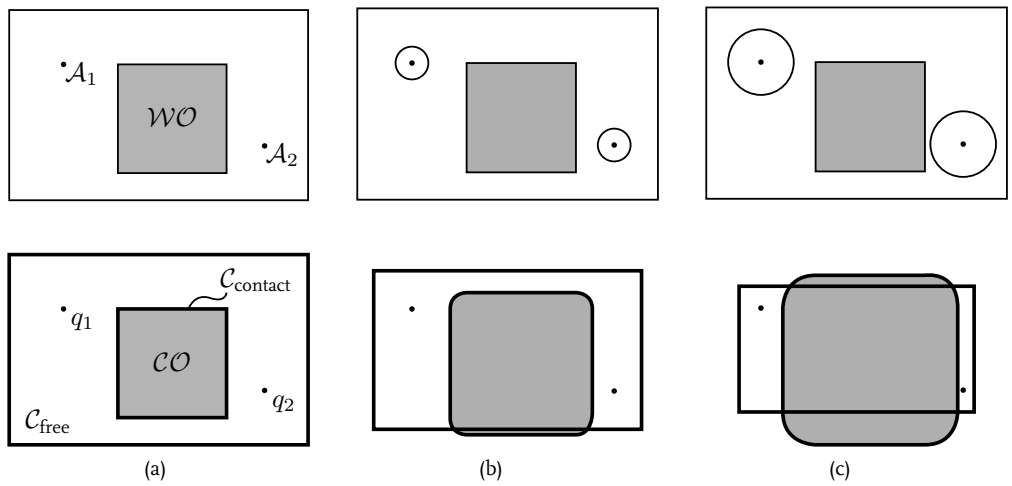


Figure 2.4: The workspace $\mathcal{W}$ (top row) and configuration space $\mathcal{C}$ (bottom row) for a robot $\mathcal{A}$ at two points, represented as a point (a), a circle (b) and a larger circle (c). $\mathcal{W}$ is populated by obstacles that make up the obstacle region, $\mathcal{WO}$. This region is mapped into $\mathcal{C}$ as the configuration space obstacle region, $\mathcal{CO}$. In subfigure (a) the contact space ($\mathcal{C}_{\text{contact}}$) and free configuration space ($\mathcal{C}_{\text{free}}$) are also visualized, as well as the two robot configurations q_1 and q_2. In subfigure (b) and (c) these annotations are omitted.

2.2.3 Construction of the Configuration Space

Lets consider the example of a mobile robot base with a circular geometry moving on a plane, so $\mathcal{W} = \mathbb{R}^2$. A configuration of this robot is described by two translations, x and y, and one rotation, θ. The robot's geometry is the same for every rotation as the geometry of the base is circular and thus the rotation is not necessary to describe a configuration. The dimension of $\mathcal{C}$ is therefore equal to that of $\mathcal{W}$, as illustrated in Figure 2.4a. $\mathcal{WO}$ is however not identical to $\mathcal{CO}$. To construct $\mathcal{CO}$ the robot is to be slid past $\mathcal{WO}$. As the center of the circular base is chosen for a configuration it suffices in this case to inflate the obstacles with the radius of the circle. The construction of configuration space for three circular geometries is shown in Figure 2.4.
Now consider the mobile base $\mathcal{A}$ is modelled as a rectangle (e.g., a car). This rectangle might still be approximated by a circle that circumscribes the rectangle. As a result $\mathcal{C}_{\text{free}}$ will be smaller and thus a path in $\mathcal{C}_{\text{free}}$ will be more conservative in terms of distance to obstacles. At some point a free path might not be available anymore as can be seen in Figure 2.4c: q_1 can not be connected to q_2. Therefore the rotation can be included in the configuration space as a third dimension. The construction of $\mathcal{CO}$ is now not so obvious anymore. $\mathcal{CO}$ has to be determined for every increment of rotation of $\mathcal{A}$ and then stacked along the axis perpendicular to the plane, as depicted in Figure 2.5. $\mathcal{CO}$ is then represented by a volume generated by orientation slices for every increment. For complex geometries and even more degrees of freedom, such as a robotic arm, $\mathcal{CO}$ is not conceivable anymore.

2.2.4 Representation of the Configuration Space

The configuration space transforms the problem of planning the motion of an arbitrarily shaped robot into the problem of planning the motion of a point. To be able to search for possible motions of that point in the configuration space, it must be represented in a way that connects configurations: the *connectivity* of the free configuration space must be captured. This representation can be of different forms. Examples of these methods to represent the connectivity of the configuration space are illustrated in Figure 2.6 for a general problem in a workspace with arbitrary obstacles.

A first way is to capture the connectivity in a network of curves that are in $\mathcal{C}_{\text{free}}$, called a *roadmap*. Another representation is a *cell decomposition*, where $\mathcal{C}$ is decomposed into discrete, non-overlapping

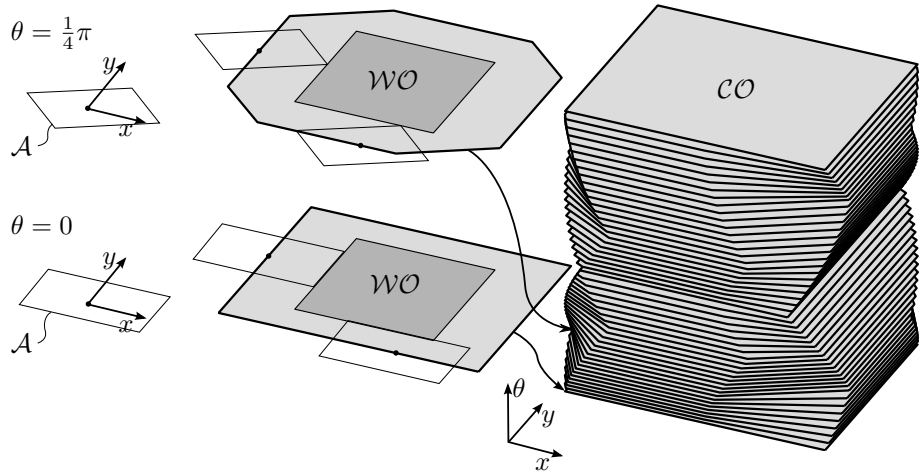
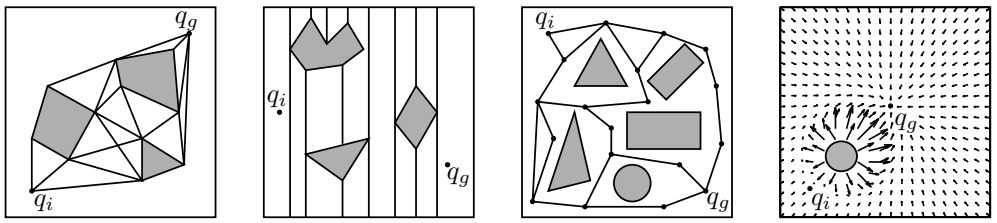


Figure 2.5: For a circular base, as in Figure 2.4, $\mathcal{CO}$ is identical for every orientation. For a non-circular base, here the rectangular shaped robot $\mathcal{A}$, $\mathcal{CO}$ depends on the orientation of $\mathcal{A}$. It is constructed by parameterizing each configuration q by $(x, y, \theta) \in \mathbb{R}^2 \times [0, 2\pi]$. At $\pi = 0$ and $\pi = \frac{1}{4}\pi$ two rotation increments of $\mathcal{A}$ are shown together with the accompanying $\mathcal{CO}$ for θ ranging from 0 to π. The representation of $\mathcal{CO}$ is a volume consisting of $\mathcal{CO}$ per increment of orientation.



(a) Roadmap (b) Cell decomposition (c) Sampling-based (d) Potential field

Figure 2.6: Different representations of the connectivity of a configuration space with arbitrary obstacles (shaded objects). The potential field method does not capture the connectivity, but the potential field resembles the 'structure' of the configuration space.

cells that are subsets of $\mathcal{C}$. The union of those cells makes up $\mathcal{C}_{\text{free}}$. A third way is to represent the space in a stochastic manner, with a *sampling-based* method. The idea behind this is to represent the connectivity of $\mathcal{C}_{\text{free}}$ without explicitly constructing the space itself. It can be noticed that the example in Figure 2.6c looks like a roadmap method. The emphasis is however on the stochastic character and therefore it is treated as a sampling-based method. A final method of representing the configuration space is using a *potential field*. The point in $\mathcal{C}$ that represents the robot then moves under the influence of a potential field obtained by superposing an attractive potential towards the goal and a repulsive potential from $\mathcal{CO}$.

Roadmap, cell decomposition and sampling-based methods capture the connectivity of $\mathcal{C}_{\text{free}}$ into an abstracted graph that can be searched for a path, as will be explained in the next section. A potential field method is based on a different idea, as it suggests that robot moves under the influence of attractions and repulsions. The local variations in the potential field reflect the 'structure' of $\mathcal{C}_{\text{free}}$. A potential field methods therefore does not need a graph search to return a path, but instead guides the robot though the workspace in the continuous world. More detailed classifications of configuration space representations are treated in Chapter 4.

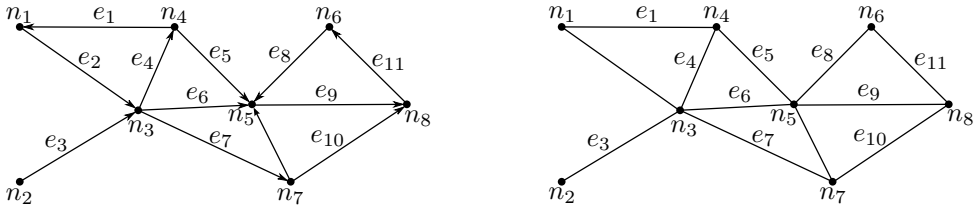


Figure 2.7: A directed graph (left) and an undirected graph (right) with nodes n and edges e.

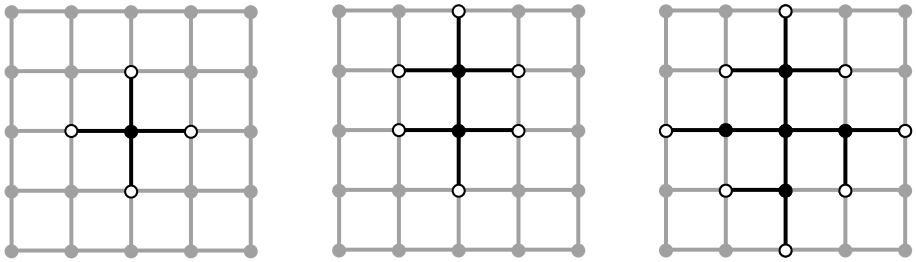


Figure 2.8: A graph search on a grid. The frontier (white nodes) separates the explored set (black nodes) from the unexplored set (grey nodes). In the first iteration the root node is expanded. Then one leaf node is expanded. Finally the remaining children nodes from the root are expanded in clockwise order.

2.3 Searching the World

Most representation methods transform the continuous problem of finding a path in $\mathcal{C}_{\text{free}}$ into a discrete problem of searching a graph. A graph is a collection of *nodes* N (also referred to as *vertices* in literature) and *edges* E, denoted as $G(N, E)$. An edge connects two nodes and therefore defines a relationship between these two nodes. This could be for example two nodes in cells that are adjacent. Edges can be directed and undirected. A graph that consists of edges that can only be traversed in one direction is a *directed graph*. An edge that connects two nodes is undirected if a robot can move back and forth on that edge and the collection of those nodes and edges is an *undirected graph*. A directed and an undirected graph are illustrated in Figure 2.7. An edge in a graph can be annotated with a non-negative value, called a *weight*, that represents the cost of traversing that edge. A graph with weights is a called a *costmap*.

A graph is searched like a tree. For a grid a *graph search* is depicted in Figure 2.8. The first node at which the search of a graph starts is called the *root*. From the root the search is expanded. A node is called a *parent* if that node has subsequent nodes that can be expanded, which are called *children*. A node that has no children (yet) is a *leaf* and a series of nodes connecting the root to a leaf is a *branch*. At every search step, a leaf node is expanded, making it a parent of the expanded children nodes. The set of all leaf nodes available for expansion at a search step is called the *frontier*. The process of expanding nodes on the frontier continues until either the goal configuration is found or there are no more nodes to expand. All graph search algorithms share this basic tree search structure. The vary primarily according to how they choose which node to expand next: the *search-strategy*. This order of expansion is determined by a function $f(n)$. Which nodes can be chosen depends on the adjacency relation. In Figure 2.8 the nodes are *4-connected*, i.e., every node has four neighbors. If the diagonal nodes are also expandable it is *8-connected*.

A graph search keeps track of the nodes it visited in a *explored set* or *closed list*. This is what distinguishes a graph search from a tree search. In a graph search the frontier separates the explored set from the unexplored set of nodes. A tree search does not remember an explored set and thus it can create redundant paths in a search. For example, in the graph of Figure 2.7 a tree search might result in paths containing loops. The behavior of tree and graph search methods can be generalized to the same basic steps as summarized in Figure 2.9.

Graph search algorithms can be divided into three categories:

function TREE_SEARCH (*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

function GRAPH_SEARCH (*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
  initialize the explored set to be empty
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

Figure 2.9: An informal description of the general tree search and graph search algorithms. The parts of GRAPH_SEARCH marked in bold italic are the additions needed to handle repeated states. This part is the difference between a TREE_SEARCH and a GRAPH_SEARCH.

- ▷ uninformed
- ▷ informed
- ▷ local search

Uninformed search algorithms move through the graph without any preference for the location of the goal node. If the direction of the goal node is known, the search can be directed towards this node. A search that includes information about the goal is called informed. To this cause a *heuristic* can be formulated. This is defined as a function of nodes that hypothesizes a cost towards the goal node. The choice for the next node to explore is then based on this heuristic cost. This could be for example the Euclidean distance towards the goal. A heuristic is applied to speed up a search. However, there is no guarantee that the path that is found is the shortest. Both uninformed and informed search algorithms explore search spaces systematically, as they keep one or more paths in memory and record which alternatives have been explored at each point along the path. If the path to the goal does not matter, a different class of algorithms might be considered, ones that do not worry about paths at all. These local search algorithms operate using a single current node (rather than multiple paths) and generally move only to neighbors of that node. Typically, the paths followed by the search are not retained. Local search methods also apply to a search in the continuous space as for the potential field method. In Chapter 5 all these different strategies are discussed in more detail.

2.4 Global Versus Local

As mentioned, roadmap, cell decomposition and sampling-based methods capture the connectivity of $\mathcal{C}_{\text{free}}$. The potential field approach does not capture the connectivity in a initial processing step, but at each instance the robot moves from one configuration to the other it computes the potential field. As

Figure 2.10: The motion problem of driving a car from A to B that can be abstracted to global planning on a map and local planning on the highway.

was illustrated in Figure 2.6d, the forces in the field that move the robot depend on obstacles that are near the robot's own configuration. Therefore, potential field methods are often referred to as *local* methods, while roadmap, cell decomposition and sampling-based methods are called *global* methods.

However, the distinction between local and global is often intuitive. E.g., a potential field method can be combined with graph searching techniques that use the whole $\mathcal{C}$, which makes them as global as any roadmap. On the other hand, a roadmap method could be restricted to a subset of $\mathcal{C}$ around the current configuration of the robot. It is then used to plan subpaths and can be regarded as a local planner.

A more formal definition of global and local is desired. A global planner is defined as a planner that can use the complete workspace to return a solution. A local planner is defined as a planner that uses a subset of that same workspace.

2.4.1 An Example: a Global and a Local Planner

Consider the problem of driving a car from Eindhoven University of Technology to Amsterdam Schiphol Airport. A human approach to this problem would likely begin with specifying some high-level subtasks:

1. Drive to highway leading out of Eindhoven.
2. Plan a route from Eindhoven to Amsterdam.
3. Drive from the incoming highway in Amsterdam to Schiphol Airport.

Low-level decisions such as on what lane to drive on the highway are not even considered until they are moments away. It does not make sense to plan this ahead as these decisions depend on the current situation on the highway. A human therefore first solves a global problem and then deals with the local problem of driving safe on the highway. Therefore, the motion problem is abstracted to a global planning problem and a local (motion) planning problem as is visualized in Figure 2.10. This level of abstraction is also used for solving motion planning problems for robots.

2.5 Extensions of the Basic Problem

The basic motion planning problem is a relaxed version of the motion planning problem. The robot is a single, rigid body that can move freely and has no dynamics. It acts in a static, known environment. Due to these assumptions the problem is simplified, limiting the practical implementations of the solutions to the problem. Therefore, to meet with the conditions of the actual problem three extensions of the basic motion planning problem are regarded:

- ▷ planning in a dynamic environment
- ▷ planning with uncertainty
- ▷ planning with constraints

The way to deal with extensions that encompass the actual problem is called a *planning approach*. A planning approach views the motion planner as a whole. So at this point the representation method and the search algorithm come together. This section discusses the subproblems of the three extended problems and introduces necessary additional terminology. How different planning approaches tackle the motion planning problem is discussed in Chapter 6.

2.5.1 Planning in a Dynamic Environment

In the basic motion planning problem the environment is considered to be completely *static* as the robot $\mathcal{A}$ is the only moving object in the environment. The environment can also be dynamic, when it contains *moving objects*. Another type of environment occurs when not only the motion of the robot $\mathcal{A}$, but of *multiple robots* $\mathcal{A}_i$ is to be planned (Erdmann and Lozano-Pérez, 1986; Latombe, 1990). This case differs from an environment with moving objects, as now the motion of more than one robots is under control. Finally, a special case arises when *manipulation* (Choset, 2005; Li et al., 1989) is considered. In this case, the ability to alter the environment during movement, by moving objects itself, must be taken into account by the motion planner. Planning for manipulation is such a broad topic in itself that it is also been addressed with techniques that are outside the scope of motion planning. This study will therefore not go into depth on this subject.

In the presence of moving obstacles, the configuration space changes over time. To solve the motion planning problem the configuration space can be extended with a time dimension. This space is called *configuration-time space* (Erdmann and Lozano-Pérez, 1986) and is denoted by $\mathcal{CT}$. $\mathcal{W}$-obstacles map in the $\mathcal{CT}$-space to static regions called $\mathcal{CT}$-obstacles. A cross-section through $\mathcal{CT}$ at time t represents the configuration space of the robot at time instance t. For $\mathcal{W} = \mathbb{R}^2$ with a piecewise linear moving obstacle $\mathcal{CT}$ and three time instances $t \in [0, T]$ are depicted in Figure 2.11. Motion planning now entails finding a path among $\mathcal{CT}$-obstacles in $\mathcal{CT}$.

2.5.2 Planning with Uncertainty

The basic motion planning problem is based on assumptions about the robot and obstacles in the workspace. It assumes exact knowledge of the workspace and the obstacles' location and geometry. Furthermore, it is assumed that the planned path is executed exactly. Such assumptions are generally not realistic and therefore uncertainty must be considered in: a priori knowledge on the workspace; in sensor information that is acquired during the execution of planning; and in the execution of the plan itself.

This problem with uncertainty is illustrated in Figure 2.12. A robot $\mathcal{A}$ has no exact position information, due to a localization error, and therefore its initial and goal configuration are now respectively an initial region $\mathcal{I}$ and a goal region $\mathcal{G}$ in $\mathcal{C}$. Due to uncertainty in execution it assumes movement along a

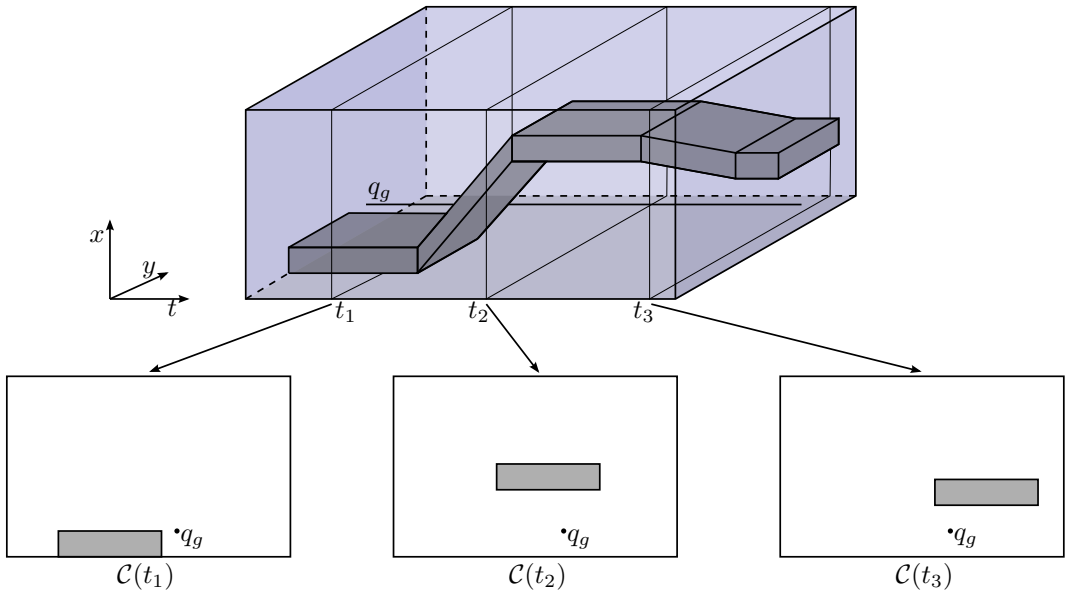


Figure 2.11: A configuration-time space, $\mathcal{CT} = \mathbb{R}^2 \times [0, T]$, for a workspace with a piecewise linear moving obstacle. On the bottom row three time instances or 'slices' of the space are depicted.

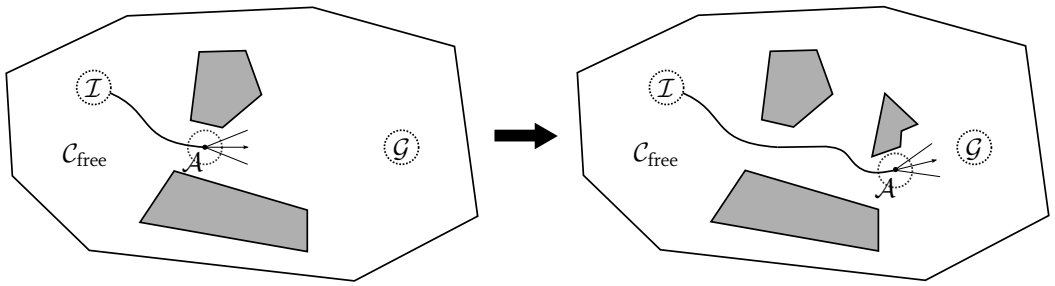
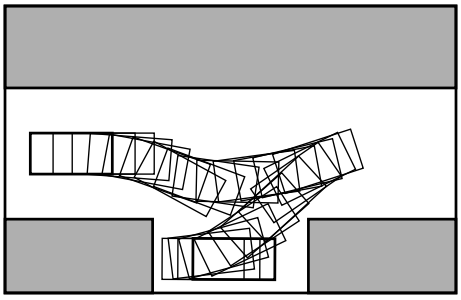


Figure 2.12: Two instances of a motion planning problem with uncertainty. The initial and goal configuration are represented by regions, respectively $\mathcal{I}$ and $\mathcal{G}$. Furthermore, robot $\mathcal{A}$ has no exact localization and moves along a direction contained in a cone centered along the commanded direction. As the robot moves along the planned path it can encounter previously unknown, static obstacles.

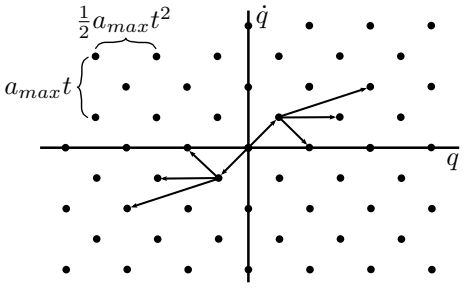
direction contained in a cone centered along the commanded direction of motion. During execution it can encounter previously unknown, static obstacles. In some literature suddenly appearing obstacles are regarded as a dynamic environment. In this study it is however regarded as uncertainty in a priori information on the workspace.

2.5.3 Planning with Constraints

The basic motion problem or path planning problem is purely geometrically constrained. For some problems however it is desired to impose additional constraints to the robot's motion. The most common are differential constraints, which restrict the motion of the system represented by the evolution of configurations $q \in \mathcal{C}$ over time. A configuration is described by a vector of generalized coordinates. Differential constraints can be considered as local constraints, on the robot, in contrast to constraints that arise due to obstacles. The solution of a motion problem is a feasible trajectory that is parameterized by time. To be feasible at each time instant differential constraints must be satisfied that arise from the kinematics and dynamics of a robot. The distinction between kinematic and dynamic constraints in the scope of this study is clarified using the configuration space. Kinematic constraints act on the space of all possible configurations of a system at one time, while dynamic constraints act on



(a) A planned motion for a car that faces the problem of parallel parking. The car is subject to a non-holonomic constraint (no-slip sideways).



(b) A motion planning problem for a double-integrator system that moves over a line with discretized bang-bang control. The system incorporates the equation of the motion and a bound on the velocity and acceleration.

Figure 2.13: An example of non-holonomic planning (left) and kinodynamic planning (right).

how configurations can change as a function of time.

Planning with only kinematic constraints is often referred to as *non-holonomic planning*. A simple example of a system in $\mathcal{W} = \mathbb{R}^2$ that is subject to a non-holonomic constraint is a car. It is able to access any configuration in $\mathcal{C}$, but it can not instantaneously move sideways as its wheels can not slip sideways. A non-holonomic constraint is a velocity constraint on $\mathcal{C}$. This constraint becomes important when a car needs to park parallel for example, as illustrated in Figure 2.13a.

Finding a trajectory that is feasible requires the robot to obey its dynamics. This includes dynamic laws and bounds on velocity, acceleration, and applied forces. This means that there are second-order constraints on $\mathcal{C}$. Planning with such constraints is known as *kinodynamic planning* (Donald et al., 1993). The state space, $\mathcal{S}$, is introduced to deal with constraints. A state s encodes a position and a velocity, $s = (q, \dot{q})$.

An example of such a constraint for a double-integrator system $\ddot{q} = a$ is illustrated in Figure 2.13b. The system is modeled as a point mass moving on a line, hence q is one-dimensional. The control is bang-bang and discretized to $\{-a_{max}, 0, a_{max}\}$. The configuration space is represented together with the set of admissible velocities. A state is represented as $s = (q, \dot{q})$. Starting at the initial state $(0, 0)$ the system can at each step either attain its velocity, accelerate or decelerate. The equation of motion limits the admissible velocity and acceleration at each time step. Furthermore the bound on the position and velocity is given by the admissible states in Figure 2.13b. The bound on the acceleration is given by the discretized control set.

The state space can also be extended with a time dimension. This approach can be used to solve motion planning problems with kinodynamic constraints and moving obstacles. The combined state $\times$ time space was introduced by Fraichard (1993) and is denoted as $\mathcal{ST}$.

Chapter 3

Requirements

In this chapter requirements are formulated to form a basis for the selection of an appropriate motion planner, that solves the problem as defined in Chapter 2. These requirements will be a guideline in the search for a motion planner that solves a motion planning problem. In Chapter 4 the different classes of representations will be discussed and compared based on the requirements that are formulated in this chapter. The same is done in Chapter 5 for search algorithms. Finally, different planning approaches are discussed in Chapter 6. In the first section of this chapter the requirements are formulated. In the second section it will be discussed how the requirements need to be interpreted for representation methods, search algorithms and planning approaches. Finally, the requirements will be formulated for the specific problem of a car that needs to navigate from one place to another, exemplifying how the requirements can be used.

3.1 Motion Planner Requirements

The simplified representation of the navigation system of a robot, introduced in Chapter 2, is shown in Figure 3.1. The Figure is the same as Figure 2.1, but the highlighting is different. The four highlighted blocks in this flowchart characterize the motion planning problem. These characteristics are the starting point to formulate requirements on a motion planner:

- ▷ High-level control: The motion planner must fulfill the specific task that is given to the robot.
- ▷ Environment: The motion planner needs to perceive an environment with different characteristics, e.g., it can be dynamic or static.
- ▷ Properties of the robot: The properties of the robot determine how it can sense the environment and how it moves in the environment.
- ▷ Properties of the motion planning method: The motion planner can solve the motion problem in many ways. For example, it can be tailored to return the fastest or shortest path.

Based upon these four characteristics the following seven requirements are formulated.

3.1.1 Task

The task that is given to the motion planner naturally implies a requirement. Three kinds of tasks are distinguished:

- ▷ navigation

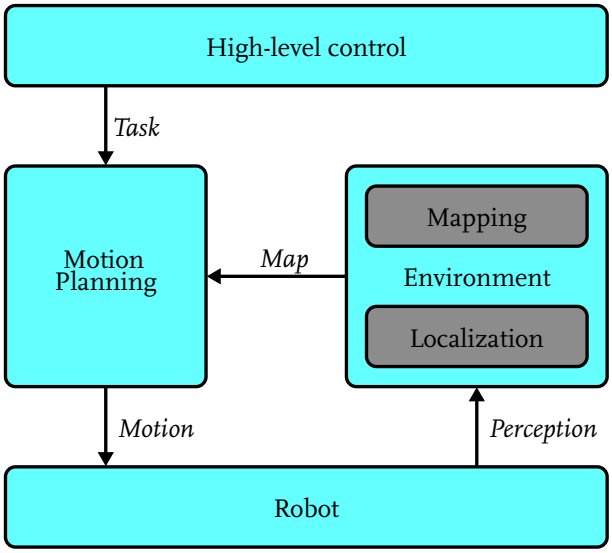


Figure 3.1: The motion planning problem is characterized by four aspects (highlighted in cyan): a task specified in the high-level control layer; the environment that is to be perceived; the properties of the robot; and the properties of the motion planner method.

- ▷ coverage
- ▷ mapping

The most general task is to navigate. *Navigation* is a very diverse term and has a variety of meanings. Generally it means 'getting from here to there'. *Coverage* of the whole environment could also be a task. Instead of moving towards one goal configuration the robot has to cover the whole map of the environment. A vacuum cleaning or lawn mowing robot are examples. *Mapping* is a task that is related to an unknown environment. A robot that has the task to map an unknown environment should explore the whole environment to be able to cover or navigate it. The idea behind mapping is that the resulting map can be used for more instances of the motion planning problem.

The far most motion planning methods aim at navigation. Coverage is less addressed in literature. Motion planners that allow coverage rely on a specific class of methods that are used for navigation. Therefore it is not taken into account as a requirement in this study. A good overview of coverage in robotics is given by Choset (2001). Mapping is necessary when the environment is completely unknown. This tends to be outside the scope of motion planning and deals more with exploration of the workspace instead. The more incomplete the prior knowledge, the less important the role of motion planning is. Therefore this is not investigated in this literature study and mapping is not considered as a requirement. For more information on mapping the reader is referred to a good overview given by Thrun et al. (2005).

This study thus only considers navigation as a task. Therefore it is not taken into account as a requirement.

3.1.2 Completeness

If a problem can be solved, one wants a problem solver that guarantees this solution, if one exists. This seems rather trivial, but in case of motion planners this is not. The guarantee of returning a solution is of course very desirable, but also very powerful in terms of computing. This requirement is known as completeness. A motion planner can be:

- ▷ complete
- ▷ resolution complete
- ▷ probabilistic complete
- ▷ incomplete

A *complete* motion planner guarantees to find a solution if one exists and reports failure otherwise. On the other hand, a planner is *incomplete* if it is not able to guarantee to find a solution if one exists. A deterministic approach that samples densely is said to be *resolution complete*. This only guarantees a solution, if one exists, at some level of resolution or discretization of the configuration space. A stochastic approach that samples densely is said to be *probabilistic complete*, meaning that if a solution exists the probability of finding a solution converges to one as the number of samples tends to infinity.

3.1.3 Optimality

The notion of optimality can be interpreted in many ways. Generally it is defined in terms of a cost, defined by a costmap. A costmap defines a cost for traversing from one configuration to another. It serves as a heuristic to the search in the configuration space. Optimality is divided into:

- ▷ optimal
- ▷ optimal only in some sense
- ▷ non-optimal

Defining an appropriate cost allows a motion planner to find motions that are *optimal* in for example the distance traveled, the energy use or the safety in distance with respect to obstacles. It is *optimal only in some sense*, if the motion planner is not optimal in any sense, but can be optimal in one of those. If there is no possibility to guarantee that the motion planner is optimal in any sense it said to be *non-optimal*.

3.1.4 Computational Complexity

The complexity of a motion planning problem depends on the complexity of the obstacle space $\mathcal{O}$ and dimension D of the configuration space. If a continuous space is approximated, the finite number of nodes used is denoted as N. This is a measure of the resolution of discretization, and also influences the complexity.

To classify the computational complexity, one is interested in proving upper and lower bounds on the minimum amount of time required by the most efficient algorithm solving a given problem. The complexity of an algorithm is usually taken to be its worst-case complexity, unless specified otherwise. To show an upper bound $T(n)$ on the time complexity of a problem for a number of inputs n, one needs to show only that there is a particular algorithm with running time at most $T(n)$. However, proving lower bounds is much more difficult, since lower bounds make a statement about all possible algorithms that solve a given problem. The phrase 'all possible algorithms' includes not just the algorithms known today, but any algorithm that might be discovered in the future. To show a lower bound of $T(n)$ for a problem requires showing that no algorithm can have a time complexity lower than $T(n)$. Providing lower bounds for motion planning methods is outside of the scope of this study. For more information on this topic the reader is referred to the work of LaValle (2006). The upper bound is still useful as it indicates the worst-case complexity of a planner.

The upper bound or worst-case complexity is typically expressed in the big O notation, which hides constant factors and smaller terms. This makes the bounds independent of the specific details of the computational model used. For instance, if $T(n) = 7n^2 + 15n + 40$, in big O notation one would write $T(n) = O(n^2)$.

3.1.5 Robustness Against a Dynamic Environment

Solving the motion planning problem in a dynamic environment is an extension of the basic motion problem, introduced in Section 2.5. A motion planner is required to be robust against this dynamic environment. A method is classified as:

- ▷ robust against a dynamic environment
- ▷ not robust against a dynamic environment.

The classification depends on the planner's ability to deal with multiple moving obstacles. A method is said to be *not robust* if the overall performance of the motion planner is less in presence of moving obstacles. This can be qualified when a planner sacrifices optimality or completeness or increases in computational complexity. A motion planner is *robust* if it deals well with moving obstacles.

3.1.6 Robustness Against Uncertainty

Just as for robustness against a dynamic environment, a motion planner is required to be robust against uncertainty that is present. A method is classified as:

- ▷ robust against uncertainty
- ▷ not robust against uncertainty

The classification depends on the planner's ability to deal with uncertainty. A method is said to be *not robust* if the overall performance of the motion planner is less in presence of uncertainty. A motion planner is *robust* if it deals well with uncertainty.

3.1.7 Dealing with Constraints

If the robot executes its path it is required to deal with the constraints that arise from the kinematics and dynamics of the robot. A motion planner is classified as:

- ▷ able to deal with constraints
- ▷ not able to deal with constraints

It is able to deal with constraints if it does not sacrifice performance in the sense of completeness or optimality or results in an increase in computational complexity. On the other hand a motion planner is said to be not able to deal with constraints if incorporating constraints is at the expense of increasing computational complexity or the sacrifice of completeness or optimality.

3.1.8 A Note on Safety

Safety is a very important requirement for motion planning and robots in general. In fact the first law of the The Three Laws of Robotics by science fiction author Isaac Asimov states: "A robot may not injure a human being or, through inaction, allow a human being to come to harm" (Asimov, 1963). If this law is translated to a motion planner it can be concluded that the planner must yield safe motions, i.e., motions that do not collide with obstacles. It is argued that safety is closely related to the requirements of being robust to a dynamic environment and uncertainty and dealing with kinodynamic constraints. For example, maintaining the speed limit on the highway (regarded as safe), is a dynamic constraint. Hence, this study does not regard safety as a separate requirement. If a motion planner satisfies the three requirements stated above it is regarded as safe.

3.2 Relevance of Requirements

The requirements of robustness against a dynamic environment, robustness against uncertainty and dealing with constraints relate to the extensions of motion planning problem, as discussed in Section 2.5. To compare representation methods (Chapter 4) and search algorithms (Chapter 5), the basic motion planning problem is used. Therefore, only the requirements of completeness, optimality and complexity are valuable to discuss for these methods. These algorithms are however the basis of methods that are able to deal with extensions of the basic motion planning problem. A planning approach, discussed in Chapter 6, deals with these extensions and so all requirements are treated. An overview of the treated requirement per chapter is given in Table 3.1

Table 3.1: An overview of which requirement is treated per chapter.

Requirement	Chapter 4	Chapter 5	Chapter 6
Completeness	✓	✓	✓
Optimality	✓	✓	✓
Computational complexity	✓	✓	✓
Robustness against a dynamic environment			✓
Robustness against uncertainty			✓
Dealing with constraints			✓

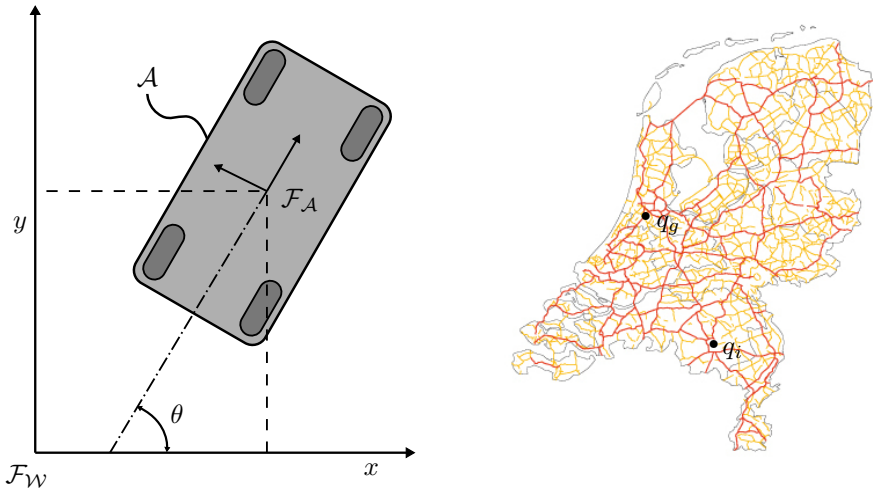
3.3 How to Use Requirements?!

How to interpret the requirements discussed in Section 3.1 depends on the specific motion problem that must be solved for a robot. To illustrate the formation of such requirements an example is considered, where a car must drive from the campus of Eindhoven University of Technology to Schiphol Airport near Amsterdam. The motion problem is now defined as finding a collision-free trajectory from the initial configuration q_i, a parking spot on the campus, to the goal configuration q_g, a parking spot at the airport. This problem will be solved in general using a GPS navigation system and a local 'human' travel planner. Hence, the motion problem is subdivided into a global path planning problem and a local motion planning problem (see also Section 2.4). Therefore there is a clear distinction between a global and a local planner. To discuss the requirements on a motion planner that solves this exemplary problem, first the workspace and configuration are defined.

The car is modeled as a rectangular object $\mathcal{A}$ that moves in $\mathcal{W} = \mathbb{R}^2$ as illustrated in Figure 3.2a. $\mathcal{A}$ is represented by three coordinates (x, y, θ), where x and y are the coordinates of the object fixed frame $\mathcal{F}_\mathcal{A}$ and $\theta \in [0, 2\pi)$ is the angle between the x-axis of $\mathcal{F}_\mathcal{A}$ and the x-axis of the workspace frame $\mathcal{F}_\mathcal{W}$. Solving the global path planning problem simplifies to the planning of a single point on a plane. Therefore the global configuration space is $\mathcal{C}_{\text{global}} = \mathbb{R}^2$. The orientation of the car is not of interest at this point. The representation of that space is illustrated by the highways (in red) and provincial roads (in yellow) in Figure 3.2b. The highways and provincial roads show how configurations within the space are connected in the form of a roadmap. Locally, on the road, the driver should be able to avoid obstacles and therefore the rotation of the car is necessary. The configuration space is then three-dimensional and described as $\mathcal{C}_{\text{local}} = \mathbb{R}^2 \times SO(2)$.

Requirements on the Motion Planner

All requirements are discussed for the motion planner as a whole, so local and global. However, it will be clear that some requirements are only meaningful to discuss for either of the planners.



(a) A car-like object $\mathcal{A}$ in $\mathcal{W} = \mathbb{R}^2$.

(b) $\mathcal{C}_{\text{global}} = \mathbb{R}^2$ and its connectivity.

Figure 3.2: The workspace and the global two-dimensional configuration space of a car that must drive from the campus of Eindhoven University of Technology to Schiphol Airport near Amsterdam. The local configuration space is three-dimensional as the orientation of the car is also necessary to plan its motion.

Completeness When navigating a car, the motion planner is required to be *complete*. At all times, the motion planner must guarantee that a motion to the airport exists, if that motion exists. If there is no possible solution to the problem, which is very unlikely, this must be reported. Such a situation can however be encountered when one forbids the planner to plan a path including a ferry crossing. If the only solution is to use the ferry crossing, the planner must report this.

Optimality When navigating a car one typically wants to get from A to B in the shortest time possible. The path planner is therefore required to be *optimal* in the sense of execution time of the generated path. This implies that some notion of time must be incorporated into the configuration space and thus more complexity. The local motion planner is generally required to generate smooth motions. Changing lanes on the highway for example is a smooth motion which is not achieved by jerking at the steering wheel.

Computational complexity A global path towards the airport should be planned as fast as possible and therefore the planner must be of low computational complexity. The path planner must deal with a $\dim(\mathcal{C}) = 2$. This path planning problem is nowadays solved within seconds by a GPS navigation system. However, it is still required to get a solution as fast as possible as a driver is generally impatient to start driving. The local motion planning is dealt with by the driver. From a driver it is required that it can react quickly to changes in the environment. It must deal with a configuration space that is far more complicated than for the global problem as it also consist of a time dimension.

Robustness against a dynamic environment The global configuration space seems static, but can change over time. In case of a traffic jam, a driver requires a quick re-plan to avoid the jam. When navigating a car the local planner is required to deal with a highly dynamic environment, containing loads of moving obstacles and static obstacles. A driver on a highway is consequently estimating the speed of near vehicles, and predicting their future motions. The planner is required to be *robust* to their behavior as these vehicles are obstacles to the car's motion.

Robustness against uncertainty The sensed information during execution of the path is subject to a great deal of uncertainty. Starting with the GPS, its localization is inaccurate in the order of meters in $\mathcal{C}_{\text{global}}$. For the local planner the human inaccuracy is in the order of decimeters in $\mathcal{C}_{\text{local}}$. Within

the local configuration space there is also inaccuracy in estimating the speed and future movements of obstacles. A driver generally becomes robust against these uncertainties by keeping more distance from obstacles. Due to the high risk of collision the motion planner is required to be *robust* to uncertainty.

Dealing with constraints Since a car is a non-holonomic platform, the driver of the car is required to deal with this constraint. Beside the non-holonomic, kinematic constraint a car is also subject to dynamic constraints. The local motion planner is required to satisfy those constraints. Due to inertia and momentum a car can not accelerate or brake infinitely fast. The driver of the car must take this into account.

Chapter 4

Representation Methods

The basic motion planning problem, introduced in Chapter 2, can be represented by a large number of methods. In Section 2.2.4 four classes of methods were introduced:

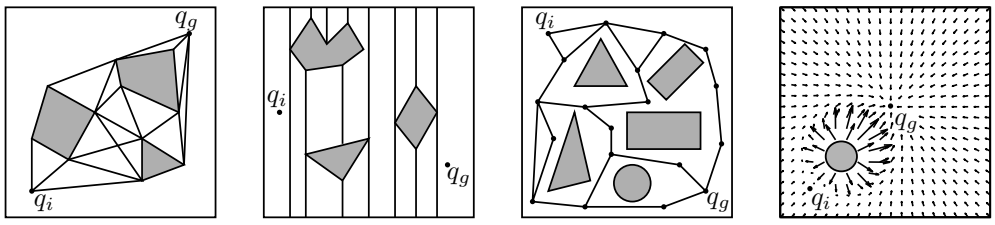
1. roadmaps
2. cell decompositions
3. sampling-based methods
4. potential fields

An example for each of these methods is illustrated once more in Figure 4.1 (same as Figure 2.6 in Section 2.2.4).

In this chapter the classes will be discussed in detail. The methods are described for a mobile robot moving in a plane, so $\mathcal{W} = \mathbb{R}^2$. To ease visualization the mobile base is represented as a point and $\mathcal{C} = \mathbb{R}^2$. For each method its properties are discussed with respect to the relevant requirements, defined in Section 3.2: completeness, optimality and computational complexity. Finally, in Section 4.5, all methods are compared to each other and conclusions will be drawn based on this comparison.

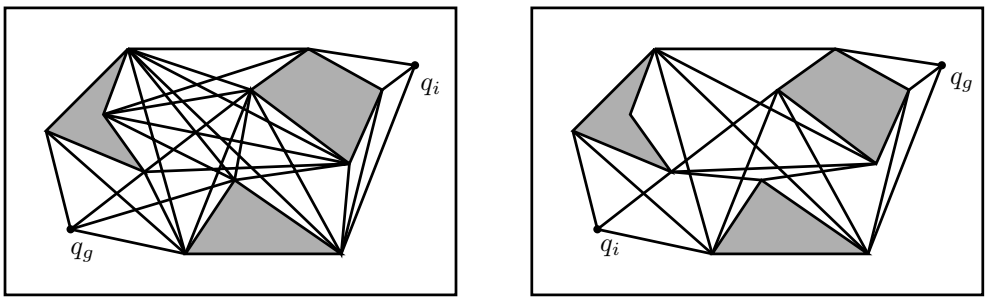
4.1 Roadmap

A roadmap is a network of curves that are in $\mathcal{C}_{\text{free}}$ and is defined as $\mathcal{R}$. The roadmap is completed by connecting an initial and goal configuration q to $\mathcal{R}$. This procedure is called retraction and is achieved by a free path, $c : [0, 1] \rightarrow \mathcal{C}_{\text{free}}$, with $c(0) = q$ and $c(1) = r(q)$, where $r(q)$ is called the retraction of q onto $\mathcal{R}$. A roadmap has to satisfy two properties:



(a) Roadmap (b) Cell decomposition (c) Sampling-based (d) Potential field

Figure 4.1: Different representations of the connectivity of a configuration space with arbitrary obstacles (shaded objects).



(a) Visibility graph. (b) Reduced visibility graph.

Figure 4.2: A visibility graph (a) and a reduced visibility graph (b) that is solely constructed from supporting and separating lines, connecting q_i to q_g.

1. Accessibility: From any configuration $q \in \mathcal{C}_{\text{free}}$ it is possible to compute a free path to a $q \in \mathcal{R}$. This condition ensures that the initial configuration q_i and goal configuration q_g can be connected to respectively $r(q_i)$ and $r(q_g)$ on $\mathcal{R}$.
2. Connectivity: If there exists a free path between q_i and q_g, then there also exists a free path between $r(q_i)$ and $r(q_g)$. This ensures that no solution is missed because $\mathcal{R}$ fails to capture the connectivity of $\mathcal{C}_{\text{free}}$.

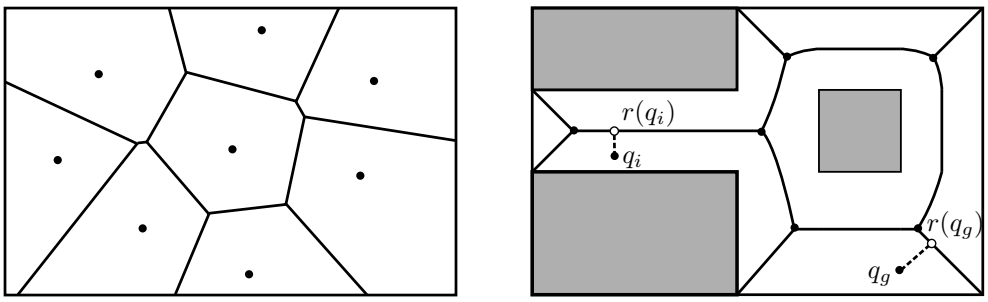
The roadmap can be constructed using different methods. Here, the two methods that are most relevant for this study are introduced: the *visibility graph* and the *deformation retract*. Other methods exist, but are only mentioned.

4.1.1 Visibility Graph

The visibility graph V is a non-directed graph. Its nodes are q_i, q_g and nodes of $\mathcal{CO}$. Two nodes of V are linked together if the line connecting the two nodes is an edge of $\mathcal{CO}$, or if that line lies entirely in $cl(\mathcal{C}_{\text{free}})$. After the construction of V it is searched for a path from q_i to q_g, and finally that path is returned or failure is reported otherwise. An example of visibility graph is illustrated in Figure 4.2a. The connectivity is represented using a few lines, but still the graph consist of many almost the same lines. Hereto, the graph can also be built solely using supporting and separating lines. A supporting line is tangent to two obstacles such that both obstacles are on the same side of that line. A separating line is also tangent to two obstacles, but such that both obstacles are on opposite sides of the line. Therefore there are four and only four tangent lines between two convex disjoint obstacles. The graph that results is a *reduced visibility graph* and depicted in Figure 4.2b.

The visibility graph has been rarely used for planning paths for $\dim(\mathcal{C}) > 2$. Higher-dimensional solutions exist, but they are at the cost of either optimality or completeness. For a robot that moves in $\mathcal{W} = \mathbb{R}^3$ with a fixed translation, so $\mathcal{C} = \mathbb{R}^3$, the generated paths may not be the shortest anymore. And a translating and rotating robot in $\mathcal{W} = \mathbb{R}^2$, with $\mathcal{C} = \mathbb{R}^2 \times SO(2)$, can be planned with a visibility graph, but it is incomplete (Lozano-Pérez and Wesley, 1979). All visibility graphs need a polygonal representation of $\mathcal{CO}$.

With respect to the requirements in Chapter 3 the visibility graph has the following characteristics:
Completeness From the connectivity property of a roadmap method it is guaranteed that a path can be found if one exists (Lozano-Pérez and Wesley, 1979), thus the method is *complete*.
Optimality As paths in a visibility graph graze obstacles the set of paths will include the shortest one (Rohnert, 1986) and therefore it is *optimal in the sense of distance traveled*. This is of course dependent on the search algorithm and distance criterion used as the shortest path still has to be returned from the graph.
Computational complexity As discussed, the visibility graph is typically only applied for $\mathcal{C} = \mathbb{R}^2$.



(a) An original Voronoi diagram, where each site is represented by a node and equidistant boundaries between two sites by edges.

(b) A generalized Voronoi diagram, where q_i and g_g are retracted onto the Voronoi diagram in $\mathcal{C}_{\text{free}}$.

Figure 4.3: Deformation retracts using an original Voronoi diagram (a) and a generalized Voronoi diagram (b).

The complexity of the visibility graph depends on the number of nodes in the graph V. A visibility graph can be computed in $O(N^2 \log N)$ time (LaValle, 2006).

4.1.2 Deformation Retracts

A deformation retract is a map that results from continuously shrinking or 'retracting' a space into a subspace. It is analogous to eroding $\mathcal{C}_{\text{free}}$ into a subspace shaped like a skeleton. This skeleton can then be used to plan the robot's motion. The most well-known method to construct such a skeleton is using a *Voronoi diagram* of $\mathcal{C}_{\text{free}}$, denoted as $\text{Vor}(\mathcal{C}_{\text{free}})$. Consider a finite set of nodes in the Euclidean plane as illustrated in Figure 4.3a. Each node is a Voronoi site, and its corresponding Voronoi cell consists of all points whose distance to this site is not greater than their distance to any other site. Each cell is obtained from the intersection of half-spaces, and hence it is a convex polygon. The edges of the Voronoi diagram are all the points in the plane that are equidistant to the two nearest sites. The Voronoi nodes are the points equidistant to three (or more) sites. For higher order site geometries the diagram turns into a *generalized Voronoi diagram* (GVD), which is depicted in Figure 4.3b. The higher order character is visible as the diagram also contains arcs besides the straight line segments. In case of a GVD the path is the product of the retractions of q_i and q_g on $\mathcal{R}$, respectively $r(q_i)$ and $r(q_g)$ and a path between these two in $\mathcal{R}$.

The concept of the GVD is also applicable in higher dimensions. The *hierarchical generalized Voronoi graph* (HVGV) (Choset and Burdick, 1995a,b, 2000) is an extension to $\mathcal{W} = \mathbb{R}^3$. This method is not considered in this study, as it is not relevant to mobile bases. Besides the Voronoi diagram, other deformation retracts methods are the *Freeway method* and the *Silhouette method*. The Freeway method is not limited to two dimensions, but it is incomplete and non-optimal. The Silhouette method is proven to be complete for an arbitrary number of dimensions with arbitrary obstacle geometries. Both have no advantage over the GVD for mobile robots and thus they are not considered. A short introduction on these other methods is given by Latombe (1990) and Choset (2005). The application of the GVD is limited to a polygonal $\mathcal{CO}$.

A deformation retract has the following characteristics:
Completeness Whatever the search strategy that is used to search the roadmap, the retraction method is *complete*, which follows from the connectivity property of a roadmap method.
Optimality Deformation retract are *optimal in the sense of distance to obstacles*. The construction of the diagram ensures a maximum distance to obstacles. This means however that it almost always excludes the shortest path in the configuration space.
Computational complexity The construction of the GVD depends on the number of edges that it contains. It runs in $O(E \log E)$ time (LaValle, 2006).

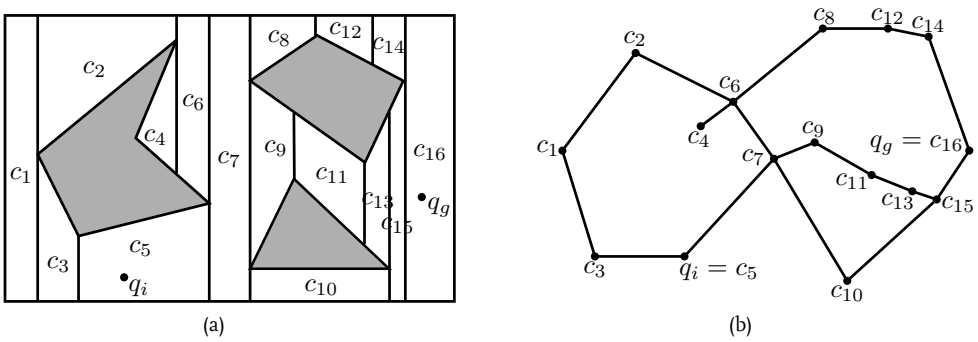


Figure 4.4: A trapezoidal decomposition using a vertical line sweep (a) and its connectivity graph (b).

4.2 Cell Decomposition

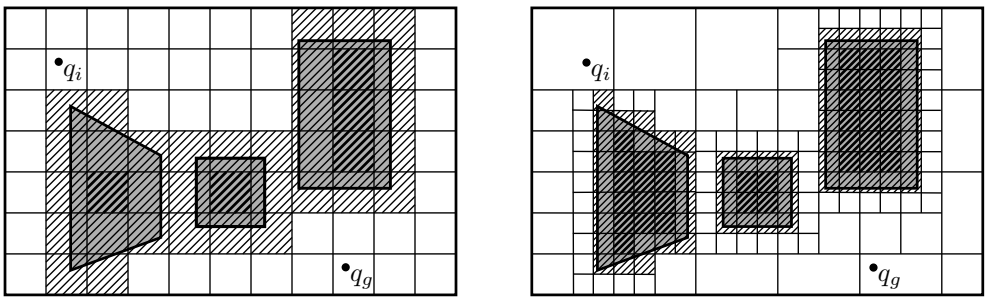
$\mathcal{C}$ can also be represented by decomposing it into discrete, non-overlapping cells that are subsets of $\mathcal{C}$ and whose union makes up $\mathcal{C}_{\text{free}}$. The shape of these cells can be arbitrary, but is such that a path between any two configurations within a cell is possible. Therefore, the cells can be represented as nodes that can be connected by edges based on an adjacency relation. The connectivity of the nodes is captured in a non-directed graph that represents the adjacency relation between the cells, called the *connectivity graph* or *adjacency graph*. By searching the connectivity graph for a path starting with the cell containing the start configuration q_i and ending with the cell containing the goal configuration q_g, a sequence of adjacent cells, called a *channel*, can be obtained, if one exists. From this channel of cells a path can be extracted that connects q_i to q_g in $\mathcal{C}_{\text{free}}$. Cell decomposition methods are further broken down into *exact* and *approximate* decomposition methods. An exact decomposition derives its name from the fact that the union of the cells of its decomposition exactly forms $\mathcal{C}_{\text{free}}$. An approximate decomposition on the other hand consists of cells of a predefined shape whose union is strictly included in $\mathcal{C}_{\text{free}}$.

4.2.1 Exact Decomposition

In an exact cell decomposition the shape and size of the cells c depends on the workspace and the location and shape of obstacles within this space. Based on the dimension of the workspace and the geometry of the obstacles multiple methods exist to decompose the robot's free space $\mathcal{C}_{\text{free}}$. The most popular cell decomposition is the vertical cell decomposition or *trapezoidal decomposition* as illustrated in Figure 4.4a. This method relies on a 2-dimensional $\mathcal{C}$ and a polygonal $\mathcal{CO}$. The *sweep line algorithm* is used to decompose $\mathcal{C}_{\text{free}}$ into trapezoids. This algorithm sweeps the configuration space in a vertical or horizontal direction with a line and makes a slice when the line detects a node of an obstacle. From the decomposition that arises, a connectivity graph C as shown in Figure 4.4b can be built, that is searched for a channel. From this channel a motion can be planned from q_i to q_g.

The extraction of a motion from the channel can be done in various ways. A common way is to select the midpoints of two joint boundaries to connect q_i to q_g.

Exact cell decompositions have the following characteristics with respect to the requirements:
Completeness An exact decomposition is a *complete* method, as $\mathcal{CO}$ is represented exact.
Optimality An exact cell decomposition method is *non-optimal*. As the cells of an exact decomposition are typically fairly large (depends on $\mathcal{CO}$) assigning a cost to a cell might not have the desired effect of an optimal path in the sense of that cost.
Computational complexity The complexity of an exact cell decomposition depends on the number of nodes in $\mathcal{CO}$. It runs in $O(N \log N)$ time (LaValle, 2006).



(a) Approximate decomposition. (b) Adaptive cell decomposition.

Figure 4.5: An approximate and adaptive cell decomposition with free (white), mixed (dashed) and occupied (bold dashed) cells. The adaptive cell decomposition uses a higher resolution in tight spaces and is therefore in this case able to resolve a sequence of free cells, whilst an approximate decomposition is not.

By recursively applying a sweeping algorithm an exact cell decomposition could essentially solve any motion planning problem, regardless of the dimension of $\mathcal{C}$ and the geometry of $\mathcal{CO}$. This method is called the *cylindrical algebraic decomposition*. It is also complete and non-optimal but the computational complexity is double exponential in the dimension of $\mathcal{C}$.

The task, introduced in Section 3.1.1, is not considered as a requirement in this study. It is however meaningful to mention it in the context of the exact cell decomposition, because this method is typically preferred when coverage of the workspace is required. This is necessary for a vacuum or lawnmower robot for example. An efficient decomposition to cover each cell is the *Morse cell decomposition* (Acar et al., 2002). More on the topic of coverage is treated in the work of Choset (2005).

4.2.2 Approximate Decomposition

Contrary to the exact cell decomposition, the approximate variant has obstacle boundaries that do not necessarily coincide with predefined cell boundaries. $\mathcal{C}$ is decomposed into a grid of cells with a predefined shape and size. Square and rectangular cells are the most dominant methods to represent $\mathcal{C}$. Rather than identifying objects or shapes, the approximate cell decomposition simply samples the workspace. The connectivity graph C is marked up accordingly to three types of cells:

- ▷ free cells, whose interior is completely within $\mathcal{C}_{\text{free}}$;
- ▷ mixed cells, that are partly in $\mathcal{C}_{\text{free}}$ and $\mathcal{CO}$;
- ▷ and occupied cells, whose interior is completely within $\mathcal{CO}$.

An example of an approximate decomposition is depicted in Figure 4.5a. A connectivity graph is built based on the decomposition with free and mixed cells as nodes, which is searched for free or mixed channels connecting q_i to q_g. The size of the cells could also be locally adapted. Such methods are referred to as *adaptive cell decompositions* and operate in a hierarchical fashion. Starting with a coarse grid, it is locally refined. The most common method is using a *quad-tree*, where mixed cells are divided into four cells. This is done in an iterative way: the algorithm divides mixed cells until a free channel is found or a minimum admissible size of cells has been reached.

The concept of an approximate decomposition is general and can be applied to a $\mathcal{C}$ of arbitrary dimension. In $\mathbb{R}^3$ for example, a tree with eight leafs called an *octree* can be used. It can deal with arbitrary shapes of $\mathcal{CO}$ as it approximates the obstacles.

The characteristics of an approximate decomposition are as following:
Completeness Figure 4.5a shows that an approximate decomposition is not complete; there is no solution, i.e., a channel of free cells, although it does exist. Figure 4.5b on the other hand, shows that when the resolution of the decomposition increases a solution arises. A planner using an approximate decomposition is therefore *resolution complete*, as it guarantees to return a solution whenever one exists for a decomposition with a small enough admissible size of cells.
Optimality An approximate cell decomposition method is *optimal*, if costs are defined for each cell based on a heuristic. It has to be remarked that the optimality depends on the resolution of the grid. Especially for an adaptive cell decomposition where large open spaces will result in large cells.
Computational complexity The construction of an approximate decomposition may seem notably simpler than an exact decomposition. It merely consists of a division of cells, followed by a collision check with $\mathcal{CO}$. However, the complexity of an approximate decomposition is exponential in the dimension of $\mathcal{C}$: $O(N^D)$. The number of nodes N depends on the resolution of the grid.

4.3 Sampling-Based Method

Representing $\mathcal{C}$ by a roadmap or an exact cell decomposition requires an explicit representation of $\mathcal{C}_{\text{free}}$. These methods become computationally cumbersome when the dimension of the configuration space increases. To decrease the complexity, the connectivity of $\mathcal{C}_{\text{free}}$ can also be constructed without explicitly constructing $\mathcal{C}$ itself. Sampling-based methods have different strategies for sampling $\mathcal{C}$ and connecting those samples to resolve the motion planning problem. A subdivision is made between *probabilistic roadmaps* and *single-query planners*. The concept of sampling-based methods is general and can be applied to a $\mathcal{C}$ of arbitrary dimension. As it is an approximate method it can also deal with arbitrary shapes of $\mathcal{CO}$.

4.3.1 Probabilistic Roadmap

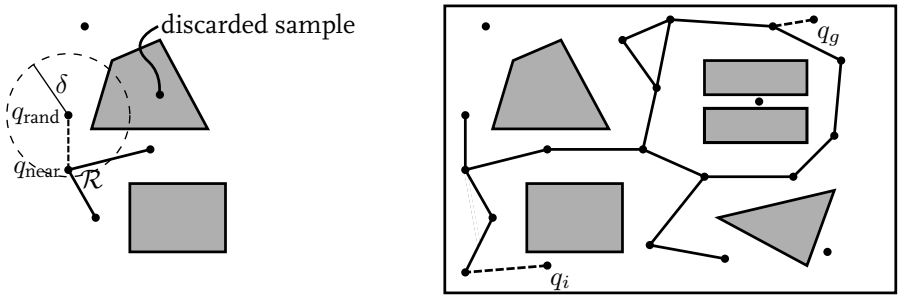
Probabilistic roadmap (PRM) methods, introduced by Kavraki et al. (1996), are closely related to the roadmap methods discussed in Section 4.1. As the name says, a PRM also constructs a roadmap, resulting in a network of edges and nodes in $\mathcal{C}_{\text{free}}$. The difference is however that a probabilistic roadmap is constructed by sampling $\mathcal{C}$ with a probability distribution, instead of explicitly constructing it based on obstacles. A sample is a node of the roadmap and represents one configuration of a robot in $\mathcal{C}$.

The construction of a PRM consists of two phases, a *learning phase* and a *query phase*.

1. Learning phase: A sampling distribution (generally a uniform probability distribution) samples a configuration q in $\mathcal{C}$. Each random sample, q_{rand}, is checked for collision with obstacles and if it is feasible it is added to the roadmap $\mathcal{R}$. Next, it is attempted to connect q_{rand} to a 'near' sample, q_{near}, according to some distance metric (generally the Euclidean distance). A typical strategy is to connect q_{rand} to q_{near} in a straight line, just as in Figure 4.6a. When successful q_{rand} and the connecting edge are added to $\mathcal{R}$. The learning phase is terminated when a predefined maximum number of sampled nodes or connecting edges is reached.

2. Query phase: During the query phase it is attempted to connect q_i and q_g to $\mathcal{R}$. If the connection is successful a path can be found by a search algorithm. If no connection can be made, because there is no solution or when $\mathcal{R}$ does not capture the connectivity of $\mathcal{C}_{\text{free}}$ sufficiently, the planner could return to its query phase or use a special strategy to enhance $\mathcal{R}$. If does not lead to success the planner returns failure.

There are many variations on this approach. These variations first of all originate from the choice for a sampling distribution. As mentioned, generally a uniform sampling approach is chosen. Examples of



(a) The learning phase of a PRM. A sample that collides with an obstacle is discarded. The uppermost sample is not connected to $\mathcal{R}$ with the distance metric δ.

(b) In the query phase it is attempted to connect q_i and q_g to the PRM. If successful the PRM can be searched for a path.

Figure 4.6: The construction of a probabilistic roadmap (PRM) consists of a learning phase (left) and a query phase (right).

other strategies are mentioned further on. Secondly, the strategy to connect q_{rand} to q_{near} can vary. In the example in Figure 4.6 a straight line was used, but other options exist. Choset (2005) references multiple discussions on the choice of this connection strategy. Different sampling distributions are also highlighted in this work.

A PRM has the following characteristics with respect to the requirements:

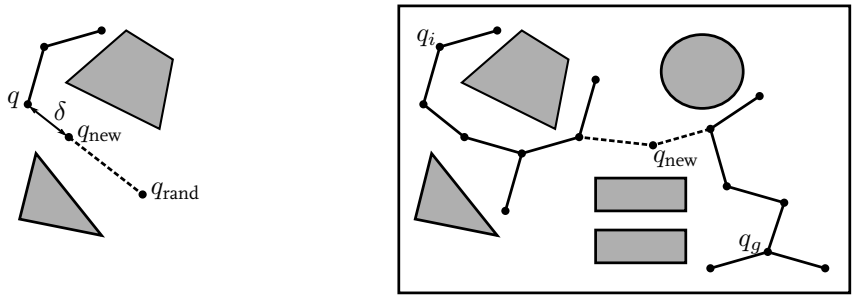
Completeness As the roadmap is stochastic a PRM is *probabilistic complete*: if a solution exists, the probability of finding a solution converges to one as the number of samples tends to infinity. The effect of the sampling distribution on the completeness is visible in Figure 4.6b. The PRM in this example is constructed using a uniform random sampling. A well-known problem is that this method has poor performance in presence of narrow passages as illustrated in the top right corner of Figure 4.6b. The uniform distribution must be very dense to connect samples inside the passage. If the only existing solution was a path through that narrow passage, the PRM is almost certainly incomplete. Hereto, different sampling strategies can be used to increase the probability of being complete. Sampling near obstacles, known as an Obstacle-Based PRM (Amato et al., 1998), is a solution. Another efficient solution is to use a Voronoi diagram of the workspace, as discussed in 4.1.2, and to sample conform to this diagram (Holleman and Kavraki, 2000).

Optimality A PRM can be informed with a cost such that a search algorithm can yield a cost optimal path, just as for the roadmaps discussed in Section 4.1. But as the roadmap is based on sampling there is a chance that this path is not even near optimality. It is concluded that a PRM is *optimal*, but with the note that this depends severely on the sampling distribution.

Computational complexity Just as for an approximate cell decomposition the complexity of a PRM depends on the level of discretization of $\mathcal{C}$. The complexity is exponential in the dimension of $\mathcal{C}$: $O(N^D)$. Typically, a PRM is of lower complexity than an approximate cell decomposition as the level of discretization is higher.

4.3.2 Single-Query Planner

Roadmaps can answer multiple queries: the same roadmap can be used to solve multiple motion planning problems in the same workspace. A single-query planner aims at quickly solving one particular instance of a motion planning problem, as they only answer one query. Instead of representing $\mathcal{C}_{\text{free}}$ exhaustively in the form of a roadmap or a cell decomposition, only a subset of $\mathcal{C}_{\text{free}}$ that is relevant to the motion problem is explored. Single-query planners explore the space with a tree structure. The tree is rooted at q_i and the planner tries to connect random, biased samples to the tree. Many different single-query planners exist, but their principle is to try to connect the random sample directly to tree. In a roadmap the new random configuration q_{rand} is added to the roadmap if $q_{\text{rand}} \in \mathcal{C}_{\text{free}}$, while in a single-query planner it is only added if it can also be connected to an existing configuration. Therefore,



(a) Adding a new configuration to a RRT. Note that only q_{new} and the edge connecting it to q are added to the tree.

(b) A bidirectional RRT rooted at q_i and q_g is merged at q_{new}.

Figure 4.7: A single-query method using a bidirectional rapidly-exploring random tree (RRT).

it follows that there is path from the root of the tree to every configuration in that tree.
It is important to realize that a single-query planner does not require an additional search to find the goal configuration.

The most used and well known single-query planner is a *rapidly-exploring random tree* (RRT). Other examples are very familiar to this approach. Its first step is to uniformly sample a random configuration, $q_{\text{rand}} \in \mathcal{C}_{\text{free}}$. Next, an attempt is made to proceed from the nearest configuration q_{near} towards q_{rand} as is illustrated in Figure 4.7a. A new candidate configuration q_{new} is produced on the line connecting q_{near} and q_{rand} at a predefined step size δ from q_{near}. This is followed by a collision check to verify that both q_{new} and the new edge are in $\mathcal{C}_{\text{free}}$. Periodically it can be checked if the tree can be connected to the goal configuration q_g, by for example inserting this configuration instead of a new random sample. Other connection strategies exist. An extensive overview of RRTs is given by LaValle and Kuffner Jr (2001).

A solution can be acquired faster by growing two RRTs, by means of a so called *bidirectional rapidly-exploring random tree*. This method uses two trees that are rooted at q_i and q_g. After a certain number of expansion steps of both trees, the algorithm enters a phase where it tries to connect the two trees by extending each one of them towards the other. An example is shown in Figure 4.7b. This type of single-query planner is not effective when q_g changes over time.

A single-query planner has the following characteristics with respect to the requirements:
Completeness The principle of a single-query planner is to explore $\mathcal{C}_{\text{free}}$ minimally. To ensure probabilistic completeness however, the sampling must be able to potentially cover the entire space. Under certain assumptions it can be shown that $\mathcal{C}_{\text{free}}$ will be covered and therefore a single-query method is *probabilistic complete* (Hsu, 2000; LaValle and Kuffner Jr, 2001).
Optimality A single-query planner is *non-optimal*. As the representation of $\mathcal{C}_{\text{free}}$ is minimal there is no guarantee that a path will be optimal in any sense. The search could be informed with a heuristic however. Typically the search is guided towards the goal and will yield a shortest path, but this cannot be guaranteed.
Computational complexity Just as a PRM, single-query planners run in $O(N^D)$ time. As the representation of $\mathcal{C}_{\text{free}}$ is typically less exhaustive, i.e., the number of nodes N lower, its computational cost lower is generally lower than a PRM or an approximate cell decomposition.

4.4 Potential Field

All methods discussed so far try to capture the global connectivity of $\mathcal{C}_{\text{free}}$, hence they are referred to as *global* methods (see Section 2.4 for the definition of global and local). Before the search for a path

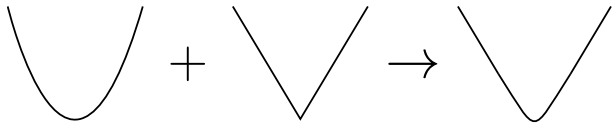


Figure 4.8: The attractive potential can be defined as a parabolic or conical function. Typically both are combined to a potential with a conical shape away from the origin and a parabolic shape near the origin.

can actually start, a precomputation step is required to construct $\mathcal{C}_{\text{free}}$. These methods are classified as *offline* methods as this precomputation generally prevents a real-time implementation.

A method that is developed for *online* motion planning is the potential field method. It does not capture the global connectivity and needs no precomputation step. Therefore it is a *local* method. Initially it was intended as a real-time obstacle avoidance method (Khatib, 1986). It treats the robot as a point in a potential field U that combines attraction to the goal, and repulsion from obstacles, as explained in Section 4.4.1, for a robot with $\mathcal{C} = \mathbb{R}^2$. The planning of that point is determined iteratively, which shows its online character. For every iteration the force $\vec{F}(q) = -\vec{\nabla}U(q)$ that is induced by the potential function at configuration q is regarded as the most promising direction of motion, and executed for some time increment. Section 4.4.2 discusses the potential field in the general case, where $\mathcal{C}$ is n-dimensional.

Local methods need powerful heuristics to guide the search as they have no global information. The drawback is that such a heuristic can lead the search to a local minimum of the potential field. The existence of local minima is explained in Section 4.4.3.

Potential field methods can be subdivided into analytical potential field methods (Section 4.4.4), numerical potential field methods (Section 4.4.5), navigation functions (Section 4.4.6), and finally grid-based navigation functions (Section 4.4.7.

4.4.1 Attractive and Repulsive Potential

The potential field U is constructed as the sum of an attractive and a repulsive potential function,

$$U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q), \tag{4.1}$$

with $U_{\text{att}}(q)$ the attractive potential towards the goal configuration q_g and $U_{\text{rep}}(q)$ the repulsive potential from the obstacles. The force $\vec{F}$ is the sum of the two negative gradients vectors

$$\vec{F}_{\text{att}} = -\vec{\nabla}U_{\text{att}}(q) \quad \text{and} \quad \vec{F}_{\text{rep}} = -\vec{\nabla}U_{\text{rep}}(q), \tag{4.2}$$

which are called the attractive and repulsive forces respectively.

The attractive potential is typically defined as a parabolic function to guide the robot to its goal, i.e.

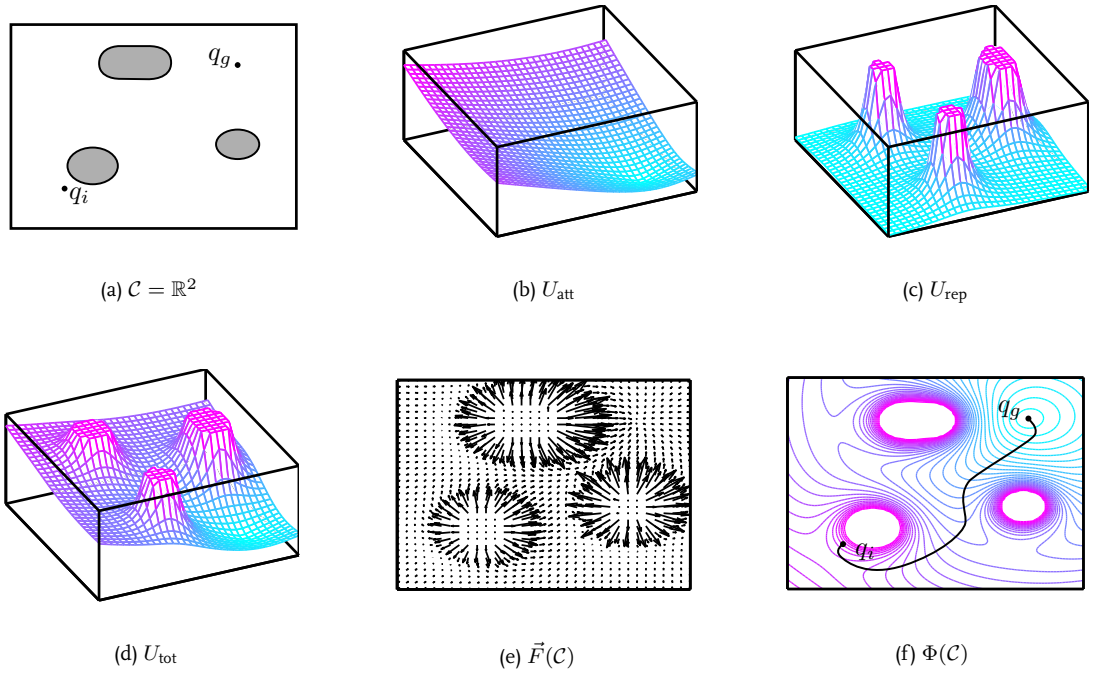
$$U_{\text{att}}(q) = \frac{1}{2}k_a d^2(q, q_g), \tag{4.3}$$

where k_a is a positive scaling factor and $d^2(q, q_g)$ is a quadratic distance criteria, typically chosen as the Euclidean distance $\|q - q_g\|$. The force this potential creates converges linearly to zero when the robot approaches q_g, but tends to infinity when $d(q - q_g) \to \infty$. An alternative is to define the attractive potential as a conical function:

$$U_{\text{att}}(q) = k_a d(q, q_g). \tag{4.4}$$

The attractive force is now constant, but indefinite in q_g. Therefore the advantages of the above two potentials are usually combined, as visible in Figure 4.8. The attractive potential is thereto defined as a conical surface away from q_g and as a parabolic surface in the vicinity of q_g.

To keep the robot away from obstacles while it is attracted towards its goal a repulsive potential is defined. The closer to the obstacle, the stronger the repulsive force should be. The main underlying



(a) $\mathcal{C} = \mathbb{R}^2$ (b) U_{att} (c) U_{rep}

(d) U_{tot} (e) $\vec{F}(\mathcal{C})$ (f) $\Phi(\mathcal{C})$

Figure 4.9: For a $\mathcal{C} = \mathbb{R}^2$ with three $\mathcal{C}$-obstacles (a), the attractive potential U_{att} towards the goal (b) and repulsive potential U_{rep} to obstacles (c), sum up to a total potential U_{tot} field as shown in Figure d. The gradient vector orientations over the field $\vec{F}(\mathcal{C})$ are displayed in Figure e. The equipotential contours of the total potential $\Phi(\mathcal{C})$ and a path that is generated by following the negative gradient of the combined potentials is depicted in Figure f.

idea is to create a potential barrier around obstacles that can not be traversed by the robot. On the other hand the robot's motion should not be influenced when it is sufficiently far away. A potential function defined for the convex components $\mathcal{CO}_i$, with $i = 1, \ldots, n_o$, of $\mathcal{CO}$ that satisfies these two requirements is

$$U_{\text{rep},i}(q) = \begin{cases} \frac{k_{r,i}}{2}\left(\frac{1}{d_i(q)} - \frac{1}{d_{0,i}}\right)^2 & \text{for } d_i(q) \leq d_{0,i}, \\ 0 & \text{for } d_i(q) > d_{0,i}, \end{cases} \tag{4.5}$$

where: $k_{r,i}$ is a positive scaling factor; $d_i(q) = \min_{q' \in \mathcal{CO}} d(q, q')$ is the minimal distance between q and $q' = \mathcal{CO}_i(q)$; and $d_{0,i}$ is a positive constant that defines to what range an obstacle influences the motion of the robot. The total repulsive potential field is then the sum of the individual potentials associated with the convex components of $\mathcal{CO}$, $U_{\text{rep}}(q) = \sum_{i=1}^{n_o} U_{\text{rep},i}(q)$. An example of a motion planning problem that is solved by potential field method is illustrated in Figure 4.9.

The total force $\vec{F}(q)$, defined as the negative gradient of the combined potentials, can be used as a feedback that guides the robot's motion towards the goal while avoiding obstacles. It can be used in three ways:

- ▷ directly as a control input τ for the robot in the form $\tau = \vec{F}(q)$;
- ▷ by interpreting the force field as a desired velocity $\dot{q} = \vec{F}(q)$ in a kinematic control scheme;
- ▷ or the robot could be considered as a mass point moving under the influence of $\ddot{q} = \vec{F}(q)$. This requires the substitution of $\ddot{q}$ in the robot's dynamic model in order to compute the generalized forces τ.

4.4.2 Potential Fields in General Case

For all methods discussed up to this point a robot is considered, without loss of generality, in $\mathcal{W} = \mathbb{R}^2$ with $\mathcal{C} = \mathbb{R}^2$. For potential field methods in the general case, where $\mathcal{C}$ is n-dimensional, the implementation is different. For the exact implementation the reader is referred to the work of Latombe (1990) or Choset (2005), but to give an idea, the general approach consist of the following steps:

1. The attractive and repulsive potential function are defined in $\mathcal{W} = \mathbb{R}^n$, with $n = 2$ or $n = 3$, instead of defining a potential in $\mathcal{C}$.
2. A number of control points is selected on the robot $\mathcal{A}$.
3. The $\mathcal{W}$-potential is computed for each of the control points on $\mathcal{A}$.
4. The $\mathcal{C}$-potential is constructed from the $\mathcal{W}$-potentials for all control points.

This approach is very useful for manipulators. A control point is defined at the end-effector (to which the goal of the motion planning problem is assigned) and at least one point for each body of the linkage. While the attractive potential only influences the end-effector control point, the repulsive potential acts on all control points. As a consequence, the potentials used in this scheme are actually twofold: an attractive-repulsive field for the end-effector, and a repulsive field for the other control points distributed on the manipulator link. The potential for each link is computed at the point where the link is closest to obstacles, and a repulsive force is applied to the link at this point. The end effector moves in the direction minimizing the sum of the obstacle potential and the goal attraction potential.

4.4.3 Local Minima

The force field $\vec{F}(q) = -\vec{\nabla}U(q)$ guides the robot's motion. This is typically implemented as a *gradient* method or an algorithm of *steepest descent* to minimize $U(q)$. The main drawback of using a potential field method is the possible occurrence of local minima; a problem that plagues all gradient descent methods. A gradient descent algorithm will converge to a minimum, where $-\vec{\nabla}U(q^*) = 0$. Such a point q^* is a *critical* point of U. This point is a minimum, a maximum or a saddle point. For a potential field with two obstacles these critical points occur as illustrated in Figure 4.10. The type of point can be determined by investigating the second order derivative, for real-valued functions, known as the *Hessian* matrix

$$H(U) = \begin{bmatrix} \frac{\partial^2 U}{\partial q_1^2} & \cdots & \frac{\partial^2 U}{\partial q_1 \partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 U}{\partial q_1 \partial q_n} & \cdots & \frac{\partial^2 U}{\partial q_n^2} \end{bmatrix}. \tag{4.6}$$

A critical point q^* at which the Hessian is non-singular, is *non-degenerate*, which implies that this point is isolated. For a positive-definite Hessian the critical point is a local minimum and for a negative-definite Hessian it is a local maximum. If the Hessian has both a negative and a positive eigenvalue at a critical point, it is a saddle point. The latter two are both unstable, as any perturbation from such a point will move the robot away from this point. A local minimum however is stable, as perturbing the robot will return the robot to this minimum. This problem is inherent to gradient methods, but it can be overcome in two ways:

- ▷ by including appropriate techniques for escaping or evading local minima;
- ▷ or in the definition of the potential function, by attempting to specify a function that has no local minima.

The two methods discussed in Section 4.4.4 and Section 4.4.5 construct potential fields with local minima and apply techniques to escape or evade these minima. A function that has no local minima is called a *navigation function* and is treated in Section 4.4.6 and Section 4.4.7.

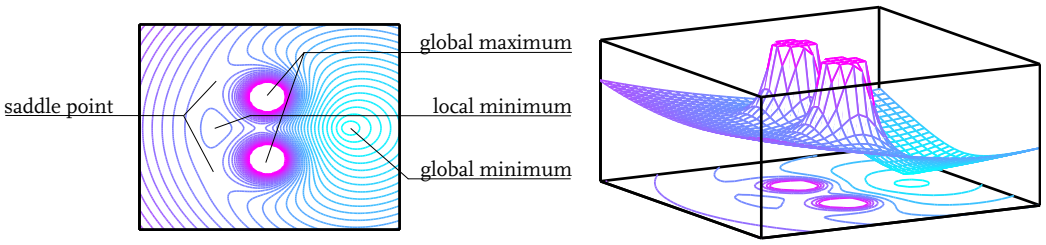


Figure 4.10: A potential field with the three types of critical points: a minimum, maximum and a saddle point. Close to the obstacles a local minimum occurs, which is a stable critical point. A robot that is close to this point will be drawn into this local minimum.

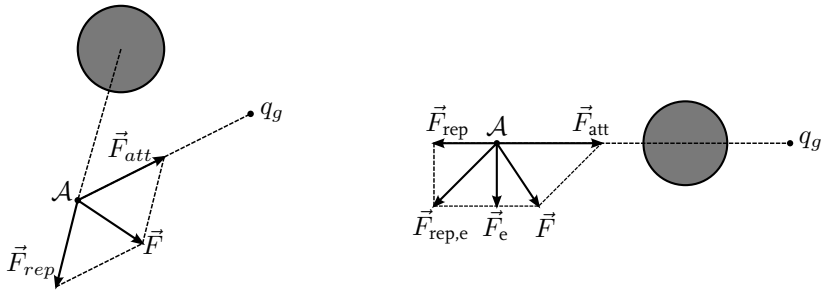
4.4.4 Analytical Potential Field Method

The most promising direction of motion is determined by $\vec{F}(q) = -\vec{\nabla}U(q)$. The determination of the gradient of the potential field requires that the attractive and repulsive force, $U_{\text{att}}(q)$ and $U_{\text{rep}}(q)$ respectively, are differentiable for every $q \in \mathcal{C}_{\text{free}}$. Analytical functions satisfy this property as they are smooth, i.e., infinitely differentiable.

An example of an implementation of an analytical potential field method is depicted in Figure 4.11. The forces in Figure 4.11a follow from (4.2), which can be analytically derived. In Equation 4.5 the shortest distance to an obstacle is chosen to determine the repulsive potential. This distance can be computed using a laser range sensor and does not require any distance calculations. However, this potential is difficult to use for asymmetric obstacles or if obstacles are partially occluded by another. In such cases the separation between an obstacle's surface and its equipotential surface can vary widely. For example, if obstacles are representable as circles with a certain radius, the potential is directly computed based on the distance measurements. If the obstacles have an arbitrary shape they are best modeled by the composition of primitive shapes such as a point, line, plane, ellipsoid, parallelepiped, cone and cylinder. This requires a description by analytic functions as described in the work of Khatib (1986), but increases complexity. Superquadric potentials (Khosla and Volpe, 1988; Volpe and Khosla, 1990) have been proposed to deal with arbitrary shapes, but this is also at the cost of increasing complexity.

The complexity of tasks that can be implemented with this approach is limited. In a cluttered environment, local minima can occur in the resultant potential field. This can lead to stable positioning of the robot before it reaches it goal. Escape or evade techniques can be used to circumvent this major issue. One of those techniques is the escape force proposed by Vadakkepat et al. (2000). When it detects that $\vec{F}_{\text{att}}$ and $\vec{F}_{\text{rep}}$ are in opposite direction it applies a perpendicular escape force $\vec{F}_{\text{e}}$, as shown in Figure 4.11b.

An analytical potential field method has the following characteristics with respect to the requirements:
Completeness A robot can get trapped in a local minimum. An escape or evade strategy can be used, but there is no guarantee that this works. Therefore an analytical potential field is *incomplete*.
Optimality It is clear that the presence of local minima will cause non-optimal motions. Methods do exist that generate smaller and less local minima, by gradually transforming the shapes of attractive and repulsive potentials. The most well-known examples are elliptical potentials (Volpe and Khosla, 1987). Such potentials however have a major drawback: obstacles must be distant from each other so that the potential of one obstacle does not influence another. The bottom line is that local minima will still be present and thus the method is *non-optimal*.
Computational complexity As the analytical potential field method is local, it is computationally much less expensive than the global approach and therefore suited for real-time implementation (Khatib, 1986).



(a) The resultant direction of the force $\vec{F}$ on the robot $\mathcal{A}$ influenced by the attractive force $\vec{F}_{\text{att}}$ and the repulsive force $\vec{F}_{\text{rep}}$.

(b) The effect of the escape force $\vec{F}_{\text{e}}$ on the direction of the resultant force $\vec{F}$ on the robot $\mathcal{A}$.

Figure 4.11: An example of resulting forces from the potential field method. A local minimum can be evaded using an escape force.

4.4.5 Numerical Potential Field Method

The potential field can also be defined over a grid of cells, i.e., using an approximate cell decomposition as introduced in Section 4.2.2. Potential field methods that rely on a cell decomposition are called *numerical potential field methods*. A search on the grid is used rather than using a gradient descent. Therefore these planner are also referred to as *search-based planners*.

A numerical potential field is a grid based version of a gradient method in the sense that the direction of movement is determined by the neighbor cell with the least cost. The cost of a cell is determined by the value of the total potential at the centroid of the cell. The descent is continued until a minimum is reached. Numerical potential field methods also suffer from local minima. Barraquand et al. (1992) shows multiple techniques to escape these minima. The most effective approach is the randomized path planner (RPP) (Barraquand et al., 1996), that uses random walks to escape a local minimum.

Contrary to the analytical potential field, the numerical methods do have a precomputation step, but its complexity is limited. The precomputation is aimed at the construction of a local minimum free $\mathcal{W}$-potential. This occurs in a space with a fixed and small size, i.e., 2- or 3-dimensional. This $\mathcal{W}$-potential acts a heuristic to construct a $\mathcal{C}$-potential that allows faster escape strategies.

A numerical potential field method has the following characteristics with respect to the requirements:
Completeness A numerical potential field method is *resolution complete* as the guarantee that a solution exists is dependent on the resolution of the grid that is used. In the case of the randomized path planner (RPP) (Barraquand et al., 1996) the planner is *resolution and probabilistic complete* because of the random strategy to escape local minima.
Optimality This method also suffers from local minima. The robot can get trapped by these minima yielding *non-optimal* motions.
Computational complexity Because the numerical potential field is based on a grid, obstacles of arbitrary size can be used. This makes the method applicable to a much wider range of problems than the analytical variant. The complexity of these methods depends on the method used to escape the local minima. The randomized path planner (RPP) is the most efficient.

4.4.6 Navigation Function

Potential fields can also be specified by functions that are free of local minima. Such a potential field is called a *navigation function* and was pioneered by Koditschek (1987). A function is said to be a navigation function in the sense of Rimon and Koditschek if it is smooth, i.e., infinitely differentiable, and has only one minimum, that occurs at the goal configuration (Koditschek and Rimon, 1990). The approach gradually transforms the shape of the attractive and repulsive potentials in order to construct

a potential field without the generation of local minima. The existence of an analytical navigation function is proved, but is only applied to a circular world with circle-shaped obstacles (Koditschek and Rimon, 1990).

Another approach is to build the potential using *harmonic functions* (Connolly et al., 1990; Connolly and Grupen, 1993). This class of functions is based on solving a partial differential equation with a Laplacian term:

$$\nabla^2 U = \sum_{i=1}^{n} \frac{\partial^2 U}{\partial q_i^2} = 0, \tag{4.7}$$

with i the number of configurations. These equations include Laplace's equation, Poisson's equation, the conduction heat flow equation, approximations to Navier-Stoke's equation, and other partial differential equations of this type. They are important in many fields of science, notably electromagnetism, fluid dynamics and heat transfer. The analogy of these fields of science and planning led to some interesting applications.

A navigation function is defined by imposing constraints on U along the boundary of $\mathcal{C}_{\text{free}}$. A *Dirichlet boundary condition* keeps the boundary at a constant value. This condition ensures that a harmonic navigation function can be developed that guides the state into a goal region from anywhere in $\mathcal{C}_{\text{free}}$. In the presence of obstacles a *Neumann boundary condition* forces the velocity vectors to be tangent to the obstacle boundary. By solving (4.7) with both boundary conditions, a harmonic navigation function can be constructed that avoids obstacles by moving parallel to their boundaries and reaches the goal.

A navigation function has the following characteristics with respect to the requirements:
Completeness A harmonic potential field that satisfies (4.7) and the Dirichlet and Neumann boundary conditions is complete (Connolly and Grupen, 1993).
Optimality The resulting path of a harmonic potential field can only be influenced by a linear combination (superposition) of the solution to the Dirichlet and Neumann boundary conditions. This allows a variation between paths that graze obstacles and paths that are repelled from obstacles. There is however no other way to specify a cost to be optimal. Therefore this method's paths are regarded as *non-optimal*.
Computational complexity The Laplace equation can be solved in two ways, numerically (Connolly et al., 1990; Connolly and Grupen, 1993; Masoud, 2010; Ryu et al., 2011) or analytically (Daily and Bevly, 2008; Feder and Slotine, 1997; Guldner and Utkin, 1993; Keymeulen and Decuyper, 1994). Numeric solutions are solved using finite-difference methods, for example Gauss-Seidel iterations as used by Connolly and Grupen (1993). The advantage of numerically solving is that any shape obstacle can be defined. Its disadvantage is the time required to numerically solve the Laplace equation, which is exponential in the dimension: $O(N^D)$.

The main advantage of an analytic solution is the fast solution for the potential field, by superposition of relatively analytical shapes. The shortcoming is that it cannot deal with arbitrary shapes.

4.4.7 Grid-Based Navigation Function

A grid-based navigation function is a method that constructs a navigation function on a grid. So in fact it is a form of an approximate cell decomposition. It is however treated as separate, because it results in a navigation function.

A grid-based navigation function is computed by a *wavefront expansion*. It starts by assigning the value 0 to goal configuration q_g. Next, its neighbor nodes are assigned with the value 1. This continuous until all cells in $\mathcal{C}_{\text{free}}$ are reached, resulting in a measure of distance from every cell to the goal. The exploration is analogous to a wave that rolls through the workspace, starting at the goal. This procedure is known as NF1 (Latombe, 1990). Using a search algorithm the shortest path to the goal can be found. A drawback of this method is that it induces paths that generally graze obstacles.

An improved grid-based navigation function, NF2 (Latombe, 1990), is formed by computing the distance from every $q \in \mathcal{C}_{\text{free}}$ to the obstacles. In two dimensions, this procedure is analogous to a

Voronoi diagram (see Section 4.1.2). This navigation function induces paths that lie as far as possible from obstacles.

For a detailed description of the NF1 and NF2 the reader is referred to the work of Latombe (1990).

A grid-based navigation function has the following characteristics with respect to the requirements:
Completeness The grid-based navigation function is *resolution complete* due to the grid representation of $\mathcal{C}$.
Optimality The definition of an attractive and repulsive potential allows a variation between paths which graze obstacle surfaces, and paths which are repelled from obstacles. Such variation can be used to optimize performance. Furthermore, this navigation function do not have local minima. This allows motions that are *optimal*.
Computational complexity The complexity of both the NF1 and NF2 procedure is $O(N + N \log N)$ (Latombe, 1990).

4.5 Conclusions

The properties of the representation methods with respect to requirements are summarized in Table 4.1. Blank table cells correspond to properties of algorithms that cannot be determined from the literature.

The requirement of completeness is very strict. Requiring completeness requires an exact representation of $\mathcal{CO}$, which proves to be difficult if obstacles are of arbitrary shape. Roadmap methods and the exact cell decomposition are typically only used for $\dim(\mathcal{C}) = 2$, because they sacrifice completeness and/or optimality for a higher order $\mathcal{C}$. Furthermore, obstacles must be represented as polygons or by analytical functions. This is at the cost of computational complexity. Weaker notions of complexity, that relate to an approximate representation of $\mathcal{C}$, have been introduced (resolution and probabilistic completeness) to reduce computational complexity and to deal with arbitrary shaped obstacles. This is achieved in two ways:

1. Approximate cell decomposition. This grid representation is a discretization of $\mathcal{C}$. It can deal with arbitrary shaped obstacles and is applicable to problems with arbitrary numbers of dimension. The complexity is however exponential in the dimension.

2. Sampling-based methods. $\mathcal{C}_{\text{free}}$ can also be constructed by random sampling of $\mathcal{C}$. Similar to a grid representation it can deal with arbitrary shaped obstacles and it is applicable to arbitrary dimensional problems.

The higher the resolution and respectively the denser the sampling, the closer the method is to completeness. Computational speed is gained by lowering the resolution or sampling density but the approximation may obscure a path (for example in Figure 4.5a). Sampling-based methods typically need less nodes to find a solution. The probabilistic completeness of sampling-based methods is dependent on the sampling distribution and strategy that is used.

A representation method determines the connectivity of $\mathcal{C}_{\text{free}}$. It can exclude possible optimal motions. Roadmap methods reduce $\mathcal{C}_{\text{free}}$ to a set of standardized paths. They can be optimal in either the sense of the shortest path (visibility graph) or in the sense of a maximum distance to obstacles (deformation retracts).
The probabilistic roadmap is an exception as its optimality depends on the sampling and strategy that is used. Just as an approximate cell decomposition it can represent $\mathcal{C}_{\text{free}}$ more extensively, without excluding possible optimal paths. By applying a cost to the nodes of these methods they can be optimal in any sense.

Roadmap, cell decomposition and sampling-based methods capture the global connectivity of $\mathcal{C}_{\text{free}}$ into a discrete graph that can be searched using a search method, introduced in Chapter 5. Potential field

methods are based on a different idea, as it suggests that robot moves under the influence of attractions and repulsions. The local variations in the potential field reflect the 'structure' of $\mathcal{C}_{\text{free}}$. The potential field was initially intended as an obstacle avoidance module, based on local sensor information only. This analytical potential field suffers from local minima and can only deal with obstacles that are described by analytical functions. The numerical variant is able to deal with arbitrary obstacles, but also suffers from local minima. Navigation functions are applied globally and create potential fields without local minima. The analytical variant has the main drawback it can only deal with analytical described obstacle shapes. The grid-based version is free of local minima and can deal with arbitrary shapes but its complexity is exponential in dimension of $\mathcal{C}$.

Table 4.1: Comparison of representation methods.

	Representation Methods									
	Roadmap		Cell Decomposition		Sampling-Based		Potential Field			
Characteristics	Visibility Graph	Deformation Retracts	Exact	Approximate	Probabilistic Roadmap	Single-query[a]	Analytical	Numerical	Navigation Function	Grid-Based
Completeness	complete	complete	complete	resolution	probabilistic	probabilistic	incomplete	resolution[d]	complete	resolution
Optimality	shortest path	distance to obstacles	non-optimal	optimal[b]	optimal[c]	non-optimal	non-optimal	non-optimal	non-optimal	optimal[b]
Computational complexity	$O(N^2 \log N)^e$	$O(E \log E)^e$	$O(N \log N)^e$	$O(N^D)$	$O(N^D)$	$O(N^D)$			$O(N^D)^f$	$O(N + N \log N)$
Number of dimensions	2	2	2	arbitrary	arbitrary	arbitrary	arbitrary	arbitrary	arbitrary	arbitrary
Obstacle region shape	polygonal	polygonal	polygonal	arbitrary	arbitrary	arbitrary	analytical	arbitrary	analytical	arbitrary

Denotations: N denotes the number of nodes; E denotes the number of edges; D is the dimension of the problem.
Superscripts: [a] already includes search algorithm [b] depends on decomposition resolution [c] depends on sampling density and strategy [d] in addition probabilistic complete for randomized path planner (RPP) [e] depends on number of obstacles [f] if solved numerically.

Chapter 5

Search Algorithms

The representation method of a space, introduced in Chapter 4 needs to be searched for a solution to the motion planning problem. Search algorithms are divided into three categories, as introduced in Section 2.3:

- ▷ uninformed
- ▷ informed or heuristic
- ▷ local search

This chapter will address the functioning of the most well-known and most used search algorithms within these three categories. These algorithms are then discussed with respect to the relevant requirements, defined in Section 3.2: completeness, optimality and computational complexity. The requirement of complexity is typically addressed by investigating the time complexity and space complexity. The former indicates how long it takes to find a solution and the latter about how much memory is needed to perform the search. For finite graph searches the worst case time complexity of most algorithms discussed in this chapter is equal. In the worst case these methods need to store every node in the search space: $O(N)$. The only exception in this chapter are local search methods. The time complexity for this method will be addressed separately. The methods with $O(N)$ time complexity can be adjusted such that they use bounded memory, without sacrificing any performance. These methods are not treated here, but for an introduction on such algorithms the reader is referred to the work of Russell and Norvig (2010).

In Section 5.1 uninformed search algorithms are discussed by means of the breadth-first and depth-first search and also Dijkstra's algorithm. Section 5.2 explains the informed or heuristic search using the greedy best-first search and the A* algorithm. In Section 5.3 local search algorithms are treated. Finally, in Section 5.4, the algorithms will be compared against each other and conclusions will be drawn based on this comparison. There exist many more algorithms than the ones treated here. The discussed algorithms however give a general impression as they form the basis of all algorithms. The terminology that is used in this chapter has been introduced in Section 2.3.

5.1 Uninformed Search

Uninformed searches cover search strategies that have no additional information about the goal. Therefore they are also sometimes referred to as *blind search* strategies. All they can do is generate successors and distinguish a goal node from a non-goal node.

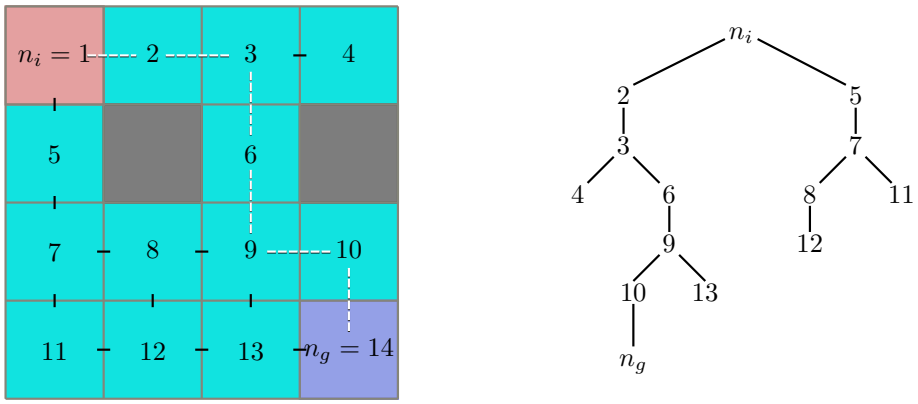


Figure 5.1: An example of a bread-first search on a 4-connected grid (left). At the beginning of the search a tree (right) is rooted at initial node n_i. The solution is found by backtracking the nodes in the tree from the goal node n_g.

5.1.1 Breadth-First

In a bread-first search the root node is expanded first, then all the successors of the root node are expanded, next their successors, and so on. At every expansion the node is tested whether it is the goal node or not, until the goal node is found. In general, all the nodes at a given depth in the search tree are expanded before any node at the next level is expanded. Consider the example of a BFS in Figure 5.1. The robot starts at n_i and can move either up, down, left or right. The search starts at the root node $n = 1$ (in pink) and begins looking for the goal by expanding all of the successors (in cyan) of the root node, node 2 and 5. These successors are not the goal, thus BFS generates each of these nodes and expands their successors, as visible in the search tree. This loop continues until the goal node $q_g = 14$ (in purple) is reached. When the goal node is reached the solution can be found by backtracking the nodes in the search tree.

The function $f(n)$ that determines the order of expansion of nodes for a BFS is

$$f(n) = g(n),$$

where $g(n)$ is determined by a FIFO (first in, first out) queue. Thus new nodes (which are always deeper than their parents) go to the back of the queue, and old nodes, which are shallower than the new nodes, get expanded first.

A BFS has the following characteristics with respect to the requirements:
Completeness A BFS will eventually find it after generating all shallower nodes (provided that the branching factor b is finite). Therefore a BFS is *complete*.
Optimality As soon as the goal node is generated, it is guaranteed that this is the shallowest goal node because all shallower nodes failed the goal test. This means that the solution will be *optimal*. Note however that the shallowest goal node is not necessarily the optimal one. BFS is only optimal if the step costs between all nodes are equal.
Time complexity In the worst case a BFS has explore every node and edge in search space and therefore its running time is $O(N + E)$.

5.1.2 Depth-First

A depth-first search (DFS) always expands the deepest node in the frontier of the search tree. Figure 5.2 shows how this search proceeds for the same example as the BFS. The search expands the same two nodes, 2 and 5, but then immediately proceeds to the deepest level of the search tree, where the nodes have no successors. This is the case at node 4. The search then 'backs up' to the next deepest node that

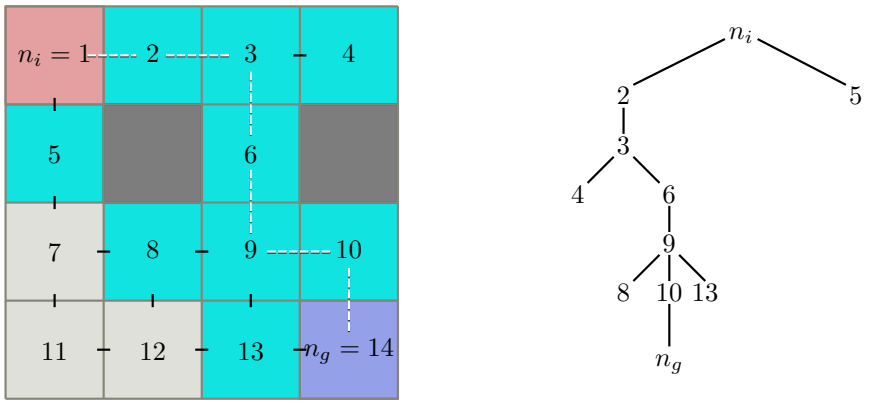


Figure 5.2: An example of a depth-first search on a 4-connected grid (left). At the beginning of the search a tree (right) is rooted at initial node n_i. The solution is found by backtracking the nodes in the tree from the goal node n_g.

still has unexplored nodes. Exploration continuous until the goal node $q_g = 14$ is discovered. Like a BFS the same solution is found but not all nodes are explored (in grey).

Whereas a BFS uses a FIFO queue, a DFS uses the same function,

$$f(n) = g(n),$$

but based on a LIFO (last-in-first-out) stack for expansion. A stack is a similar structure to the queue of the BFS. It contains the list of expanded nodes. The most recent expanded node is put at the beginning of the LIFO stack. The next node to be expanded is then taken from the beginning of the stack and all of its successors are added to the stack.

A DFS has the following characteristics with respect to the requirements:
Completeness A DFS could continue down an unbounded branch forever even if the goal is not located on that branch. Therefore it only *complete* if the search space is finite. A well-known technique that stops a DFS from continuing down an infinite branch is called *iterative deepening* which sets a limit for the depth that a DFS will search down a branch before switching to another node. This approach is the preferred uninformed search method when there is a large search space and the depth of the solution is not known.
Optimality A path is returned as soon as the goal is reached, however, this path is not always the shortest path but a path generated by the result of going down a long branch. If in this case of the example in Figure 5.2 the goal was at node 5, the resulting path would have been $2-3-6-9-8-7-5$, which is far from optimal.
Time complexity Similar to a BFS in the worst case the DFS has to explore every node and edge in search space and therefore its running time is $O(N + E)$. The complexity however severely depends on the order in which nodes are explored. Consider the example once more, where the goal is located at node 5 again. The algorithm tries to explore nodes in clockwise order starting with the node in northern direction. Thus the first node that is explored is node 2. If it starts by exploring the southern node first it would have directly found the goal.

5.1.3 Dijkstra's Algorithm

When all steps costs are equal a BFS is optimal because it always expands the shallowest unexpanded node. Dijkstra's algorithm is a method that finds an optimal solution for any step cost function. Dijkstra's algorithms expands the nodes n also using the same function as a BFS and a DFS:

$$f(n) = g(n).$$

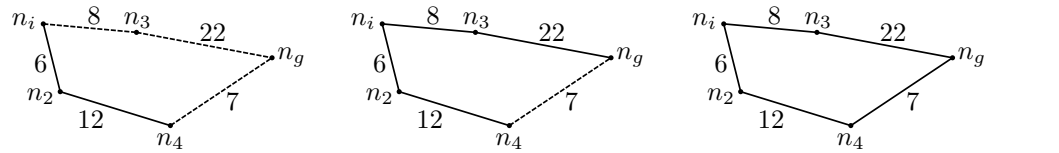
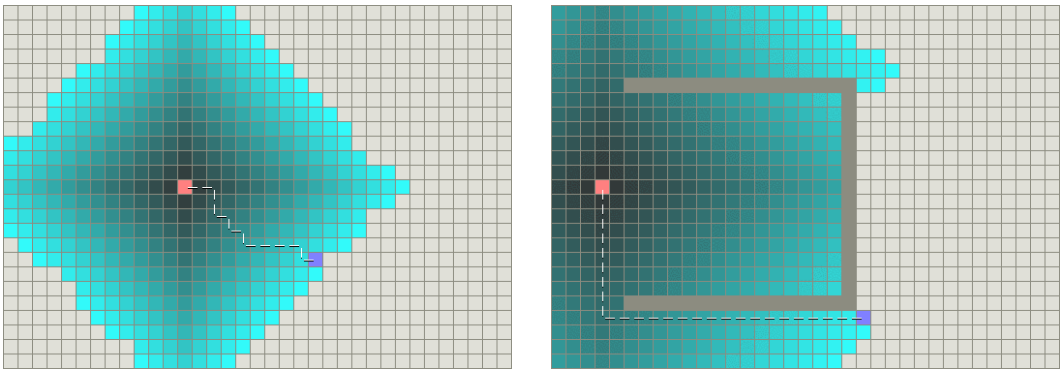


Figure 5.3: A Dijkstra algorithm always expands the node with the lowest cost first.



(a) A search without an obstacle

(b) A search with a concave obstacle

Figure 5.4: Two examples of Dijkstra's algorithm on a 4-connected grid. The pink cell is the start, the purple is the goal and the cyan cells represent the explored area.

But instead of expanding the shallowest node, it expands the one with the lowest path cost. This is done by storing the frontier as a *priority queue*, ordered by g. A priority queue can by implemented in such a way that the cost of inserting nodes, sorting them and popping nodes off the queue is $O(\log(Q))$, where Q is a measure of nodes in the queue (which is N worst case). The algorithm was initially intended to find the path with the lowest cost (i.e., the shortest path) between the root node and every other node. It can also be used for finding costs of shortest paths from a root node to a goal node by stopping the algorithm once the shortest path to the destination node has been determined. A significant difference with the BFS is that a test is added in case a better path is found to a node currently in the frontier. The results comes into play when considering the example depicted in Figure 5.3, where the problem is to get from n_i to n_g. Node n_2 is expanded first as this node has a lower cost than n_3. Next n_4 is added with cost $6 + 12 = 18$. Now node n_3 is the least-cost node, and so n_5 is added with a cost of $8 + 22 = 30$. This is the goal node, but Dijkstra's algorithm keeps exploring, by expanding n_4. This adds a second path to the goal node with cost $6 + 12 + 7 = 25$. The algorithm now checks whether this path is better and returns the solution $n_i - n_2 - n_4 - n_g$. Now consider an example on a grid in Figure 5.4. Just as for the examples of a BFS and a DFS, the pink cell is the starting point, the purple cell is the goal, and the cyan tiles show what areas are explored. The lightest cyan areas are the farthest from the starting point, and thus form the frontier of exploration. In this example the step cost are chosen equal to 1. The exploration is now equal to a BFS, as is visible in Figure 5.4a. The frontier expands like a ripple until it reaches the goal node. At this point the algorithm is terminated. In case of a concave shaped obstacle, as in Figure 5.4b, Dijkstra's algorithm needs to explore a large amount of the workspace but succeeds to find to the shortest path.

Dijkstra's algorithm has the following characteristics with respect to the requirements:
Completeness A Dijkstra search does not care about the number of steps a path has, but only about their total cost. Therefore, it could get stuck in an infinite loop if there is a path with an infinite sequence of zero-cost nodes. But if the search graph is finite, the search will eventually cover all nodes and thus it is *complete*. For infinite graphs completeness can be guaranteed by ensuring that the cost of every step exceeds some small positive constant η.

Optimality Nodes are expanded in order of their optimal path cost. Hence, the first goal node selected for expansion must be the *optimal* solution.
Time complexity The queue of Dijkstra's algorithm needs to be sorted based on the priority of nodes. The time complexity is therefore greater than those of a BFS and DFS. The priority queue is typically implemented using a *heap*. A heap is a specific data structure that is very efficient for priority queues as it orders the nodes by their cost. A Dijkstra's algorithm using a so-called Fibonacci heap has the lowest complexity known to date: $O(N \log N + E)$ (Barbehenn, 1998).

5.2 Informed Search

The uniformed search has no information about the goal. Strategies that know whether one non-goal node is 'more promising' than the other are called *informed search* or *heuristic search* strategies.

5.2.1 Greedy Best-First

Greedy best-first search tries to expand the node that is closest to the goal. Therefore it evaluates the nodes based on a heuristic, i.e.,

$$f(n) = h(n).$$

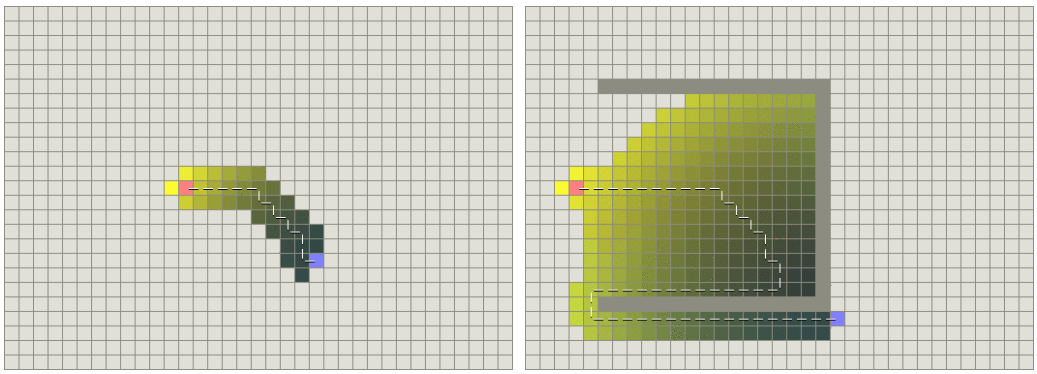
Contrary to the uninformed search algorithms a greedy best-first search does not work with $g(n)$, a cost to reach a node. It is expected that a solution is found faster solely using the heuristic. In the case of the example in Figure 5.3 a heuristic could be used that estimates the straight line distance from every node to the goal node n_g. Dijkstra's algorithm first expands n_2 because it has a lower cost than n_3. A greedy best-first search would first expand n_3 because this node is closer to n_g than n_2. At each expansion step it tries to get as close to goal as possible. Therefore it called greedy.

Now consider the same grid example as for Dijkstra's algorithm in Figure 5.5. Yellow represents those nodes with a high heuristic value (high cost to get to the goal) and black represents nodes with a low heuristic value (low cost to get to the goal). Figure 5.5a shows that a greedy best-first search can find paths exploring less space compared to Dijkstra's algorithm. However, this example illustrates the simplest case, when the map has no obstacles and the shortest path is a straight line. In case of the concave obstacle in Figure 5.5b the greedy best-first search explores less space than Dijkstra's algorithm, but its path is clearly longer. Since a greedy best-first search only considers the cost to get to the goal and ignores the cost of the path so far, it keeps going even if the path it is on has become really long.

A greedy best-first search has the following characteristics with respect to the requirements:
Completeness A greedy best-first search will eventually find the goal as long as the graph to be searched is finite. Consider the example in Figure 5.5b: the search seems trapped, but it continuous exploring even if the nodes are further away from the goal node.
Optimality It is clear that as a greedy best-first search tries to get as close to the goal as it can, it is *non-optimal*. This becomes clear for the concave obstacle example in Figure 5.5b.
Time complexity For a greedy best-first search, the priority queue is sorted by a different function than for Dijkstra's algorithm. The complexity remains the same however, $O(N \log N + E)$, depends severely on the heuristic. In workspaces with no obstacles for example the greedy best-first search will explore a minimum of nodes yielding a low computational complexity.

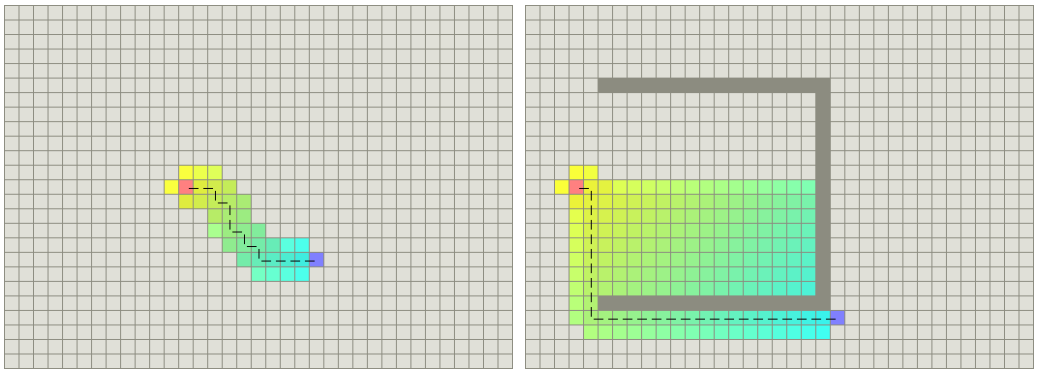
5.2.2 A*

The most popular choice for search algorithms is A* (Hart et al., 1968). Like Dijkstra's algorithm it can be used to find a shortest path. Like BFS it can use a heuristic to guide itself. It combines the cost



(a) A search without an obstacle (b) A search with a concave obstacle

Figure 5.5: Two examples of a greedy best-first on a 4-connected grid. The pink cell is the start, the purple is the goal and the cyan cells represent the explored area.



(a) A search without an obstacle (b) A search with a concave obstacle

Figure 5.6: Two examples of an A* search on a 4-connected grid. The pink cell is the start, the purple is the goal and the cyan cells represent the explored area.

function to reach a node, $g(n)$, and the cost function to get from the node to the goal, $h(n)$:

$$f(n) = g(n) + h(n).$$

By expanding the node with lowest value of $g(n) + h(n)$ the A* algorithm finds the solution with the lowest cost. Figure 5.6 shows the same example as for Dijkstra's algorithm and BFS. The heuristic cost $h(n)$ is represent by yellow, showing nodes far from the goal. The cost to reach a node $g(n)$ is represent by cyan and shows nodes far from the starting point. A* balances the two as it moves from the starting point to the goal. For the case with no obstacles, depicted in Figure 5.6a, the A* performs equal to a BFS. For the case with concave obstacle in Figure 5.6b however it finds the same path as Dijkstra's algorithm, but with only exploring less than half of the nodes. Because A* also takes into account the cost to reach each node, it does not get trapped within the concave obstacle as a BFS.

Completeness An A* search will always find a solution if that solution exists. It keeps expanding nodes until the goal node is reached, making it *complete*.

Optimality The search expands the node with lowest cost $g(n) + h(n)$. This strategy is not only complete, but if the heuristic function $h(n)$ satisfies the *consistency* condition it is also *optimal*. A heuristic $h(n)$ is consistent for every node n and every successor n' of that node if the estimated cost c of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n':

$$h(n) \leq c(n \to n') + h(n').$$

This resembles the triangle inequality, which states that each side of a triangle cannot be longer than the sum of the other two sides. The triangle is formed by n, n' and the goal node n_g.
Time complexity The A* search algorithm works in exactly the same way as Dijkstra's algorithm. The only difference is the function used to sort the priority queue. The time complexity is equal to that of Dijkstra's algorithm, i.e., $O(N \log N + E)$. Just as for the greedy best-first search the complexity depends severely on the heuristic. Worst-case the complexity is higher than that of a BFS or a DFS, but on average the complexity can be several orders of magnitude lower. For information on the effect of the heuristic on the performance the reader is referred to the work of Russell and Norvig (2010).

5.3 Local Search

Uninformed search methods have no information on the goal node. Informed search methods may have access to a heuristic function $h(n)$ that estimates the cost of a solution from node n. Both uninformed and informed search algorithms explore the search space systematically. This is achieved by keeping one or more paths in memory and by recording which alternatives have been explored at each point along the path.

If the path to the goal does not matter, a different class of algorithms can be considered, one that do not worry about paths at all. Local search algorithms operate using a single current node (rather than multiple paths) and generally move only to neighbors of that node. Typically, the paths followed by the search are not retained.

In order for a local search to reach the goal, the workspace is required to be represented as a potential field. The potential field reflects the structure of the workspace such that a search is guided towards the goal. As mentioned in Section 4.4, the potential field can be defined as a continuous and a discrete space. Local search methods can be implemented for both cases. In the discrete version the neighboring node with the lowest cost is selected for expansion. In the continuous version the most promising direction of exploration is determined by the gradient as discussed in Section 4.4.

A local search has the following characteristics with respect to the requirements:
Completeness Local search algorithms often fail to find a goal when one exists as they can get stuck at local minima. It is only complete if the goal is the only local minimum in the search space.
Optimality Because a local search does not retain any path it is *non-optimal*. It does not know whether one path is more optimal than the other.
Time complexity The time complexity of a local search is $O(1)$ as the number of nodes that is considered is constant.
Space complexity Because no path is retained only a constant number of nodes is in memory, hence a constant space complexity: $O(1)$.

5.4 Conclusions

The properties of the search algorithms with respect to the requirements are summarized in Table 5.1. A breadth-first search (BFS) and depth-first search (DFS) are the two most basic search algorithms. A BFS is preferred over a DFS as it is optimal for constant step costs. If the step costs are not equal Dijkstra's algorithm is advised, as this guarantees optimality. Its time complexity is higher, because it is implemented with a priority queue for expansion of the lowest cost node.

By informing the search with a heuristic the space and time complexity can be reduced. The most-used heuristic is the straight line distance to the goal. A greedy best-first search expands nodes with the lowest heuristic cost only. This is very effective if no obstacles are present, but in case of concave obstacles it can get trapped easily (see example in Figure 5.5b), thus it is not optimal. An A* search expands nodes based on a heuristic and the step costs as well. If the heuristic is consistent it is optimal.

Table 5.1: A comparison for graph search algorithms on a finite graph.

	Completeness	Optimality	Time complexity[1]	Space complexity[1]
Uninformed search				
Breadth-first	yes	yes[2]	$O(N+E)$	$O(N)$
Depth-first	yes[3]	no	$O(N+E)$	$O(N)$
Dijkstra's algorithm	yes[4]	yes	$O(\log(Q)*(N+E))$	$O(N)$
Informed search				
Greedy Best-first	yes	no	$O(\log(Q)*(N+E))$	$O(N)$
A*	yes	yes[5]	$O(\log(Q)*(N+E))$	$O(N)$
Local search				
Steepest-descent	no[6]	no	$O(1)$	$O(1)$

Denotations: N denotes the number of node; E denotes the number of edges; Q is the average size of the priority queue. Superscripts: [1] bounded by size of state space; [2] optimal only for equal step costs; [3] only for a finite search space [4] complete if step costs $\geq \epsilon$ for positive ϵ; [5] only if heuristic $h(n)$ is consistent; [6] complete if search space has no local minima.

Generally it is concluded that if there is a criterion for selecting a good moving direction, then informed searches are preferred over uninformed searches. The A* search is the most applied search algorithm in literature as it is complete and optimal. Worst-case its complexity is higher than those of a BFS or DFS, but on average the complexity can be several orders of magnitude lower.

If the space complexity of a method is prohibitive, variants of the discussed algorithms can be implemented that use limited amounts of memory. An introduction to such algorithms can be found in the work of Russell and Norvig (2010).

Local search can be applied if the search space is represented as a potential field. Due to the local character it has a constant complexity, but it can get trapped in local minima.

Chapter 6

Planning Approaches

The representation methods in Chapter 4 and the search algorithms in Chapter 5 deal with the basic motion planning problem as introduced in Chapter 2. A motion planning problem generally requires a robot to deal with a dynamic environment, uncertainty and kinodynamic constraints, as defined in Section 2.5. These three requirements can be met by a planning approach. A planning approach adapts the previously introduced representation methods and search algorithms to deal with the extensions of the basic problem.

A planning approach can tackle each of the three requirements separately, but it can also tackle two or even three requirements at the same time. This is visualized in Figure 6.1, where there is overlap in the three requirements. Section 6.1 starts with approaches to deal with kinodynamic constraints on the robot. This is necessary to actually execute a path. Next, approaches to deal with the robustness against a dynamic environment are considered in Section 6.2. It will become clear that there are similarities with the methods that deal with constraints, as visualized by the overlapping part in Figure 6.1. Robustness against uncertainty will be addressed in Section 6.3. The methods discussed here have overlap with the two previous sections. A class of methods that tackles all three requirements in one approach is discussed in Section 6.4. Next, other methods and issues are treated in Section 6.5. Finally, in Section 6.6 the discussed planning approaches are compared and conclusions are drawn based upon this comparison. The benchmark is a motion planning problem of a mobile robot that needs to execute a motion that satisfies kinodynamic constraints in an environment with moving obstacles and uncertainty in priori information on the workspace, sensing and execution. For each approach its effect on the completeness, optimality and computational complexity of the motion planner is indicated as either positive (+) or negative (−).

6.1 Dealing with Constraints

Kinodynamic constraints, introduced in Section 2.5.3, can be dealt with in a direct way or a decoupled way. Direct methods incorporate the constraints directly in the search for a trajectory, while decoupled methods start with a collision free path that ignores constraints and then reform this path to obtain a trajectory.

6.1.1 Decoupled Trajectory Planning

Methods that decouple path planning and dealing with constraints typically consist of three steps:

1. Path planning: Start with a given collision-free path c in $\mathcal{C}_{\text{free}}$.

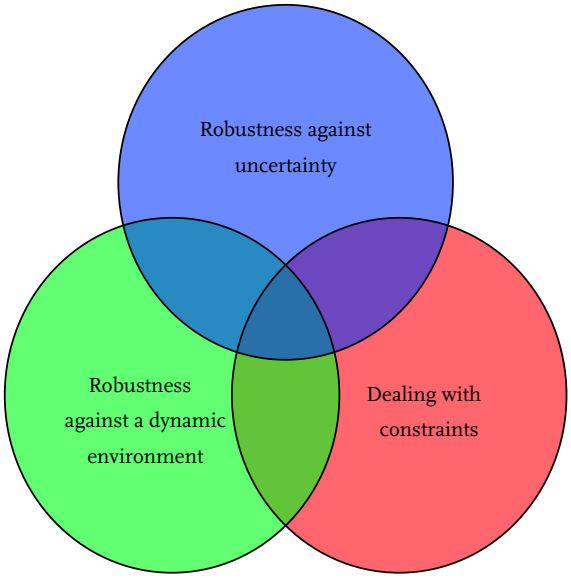


Figure 6.1: A motion planning approach must satisfy three requirements: robustness against uncertainty and a dynamic environment and it must deal with robot constraints. A planning approach can tackle these three decoupled (non-overlapping parts) but also in a coupled way (overlapping parts).

2. Plan and transform: Transform c into a new path c' to ensure that non-holonomic constraints, i.e., velocity constraints on $\mathcal{C}$, are satisfied.

3. Path-constrained trajectory planning: Compute a timing function that parameterizes c' with time, such that it satisfies the kinodynamic constraints.

The path c results from any representation method from Chapter 4 combined with a search algorithm as discussed in Chapter 5. It is transformed into c' to ensure that the robot can actually follow this path. For holonomic bases such as the TURTLEs and AMIGO this step is not necessary. In the third step c' is transformed into a trajectory, i.e., a path that is parameterized by time.

A decoupled approach divides the problem into parts that are computationally less complex to solve than the whole problem. Solving each step constitutes a solution to the problem. However, as the decomposition can introduce a problem in a preliminary step that cannot be solved by the following steps, completeness can be lost. This is especially the case for non-holonomic constrained robots, e.g., a very sharp turn that can not be made by a car. Furthermore, optimality in the sense of time is sacrificed. A path c can be made time optimal while satisfying kinodynamic constraints, but it is not guaranteed that this is the global time optimal path. This is inherent to the decoupled approach.

6.1.2 Direct Trajectory Planning

Direct trajectory planning methods satisfy kinodynamic constraints directly in the search for a solution. Nodes are added to the search graph by selecting a control input from the admissible set of controls. It is then integrated over a duration of time and it is checked for collision. The edge to this newly added node yields a trajectory from the expanded node to the new node. The order in which nodes are expanded can be determined by search methods as introduced in Chapter 5. The creation of an edge can be done using various strategies that are analogous to representation methods discussed in Chapter 4.
Another choice is to relax the restrictions on control and time and use operations over discrete stages with fixed start and end states instead. These stages, which are feasible time-parameterized curves in the state space, are called *motion primitives*. This approach is typically used for car-like robots, where the steering angle is discretized and simulated with a fixed control input over a fixed time interval.

Three important methods for planning directly with constraints are searching on a lattice, RRT-based methods and PRM-based methods. The lattice search (LaValle, 2006; Pivtoraiko et al., 2009) is analogous to performing a search on a grid acquired by an approximate cell decomposition. An equally spaced grid is obtained by choosing a control input from a discretized set and a fixed time interval δt over which the equations of motion are integrated. The double integrator system in Figure 2.13b is an example of a lattice search.

For sampling-based methods such as the RRT (LaValle and Kuffner Jr, 2001) and the PRM (Hsu et al., 2002) the control input is selected at random from the set of admissible controls. The control system is then integrated with this input over a (possibly random) time interval δt, from a previously generated node.

Exact methods such as roadmaps and the exact cell decomposition are not suited for direct trajectory planning. As their construction is dependent on obstacles, it is difficult to incorporate the kinodynamic constraints that act on the robot.

6.2 Robustness Against a Dynamic Environment

To what extent a method must be robust against a dynamic environment depends on how dynamic the environment is. A further subdivision is made between obstacles with either known or unknown trajectories. Obstacles with trajectories that are not known are regarded as an uncertainty and robustness against it is treated in the next section.

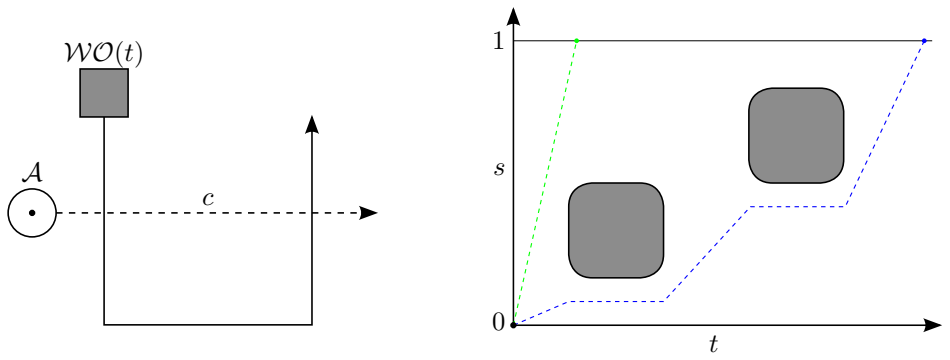
When the future locations of moving robots are known, the two common approaches are to add a time dimension to the configuration space, or to separate the spatial and temporal planning problems. This decoupling also arises when dealing with constraints. In fact, there is a major overlap in dealing with constraints and being robust against a dynamic environment. This is due the fact that a trajectory needs to satisfy dynamic constraints and must avoid obstacles.

6.2.1 Motion-Timing

Similar to the decoupled trajectory planning, discussed in Section 6.1.2, robustness against a dynamic environment can be achieved by decoupling the problem in a path planning part and motion timing part (Kant and Zucker, 1986). This approach follows the first two steps of the decoupled trajectory planning approach, but the trajectory that is formed in the third step also has to account for moving obstacles. To deal with obstacles that move over time the state space is extended with a time dimension to form a state $\times$ time space (Fraichard, 1993), introduced in Section 2.5.3 as $\mathcal{ST}$. As an example consider the circular robot $\mathcal{A}$ in Figure 6.2. A straight path c is planned, but it is intersected twice by an obstacle, as is visible in Figure 6.2a. In Figure 6.2b $\mathcal{ST}$ is shown, in which a state s indicates the time t and the position along the path, $q \in [0, 1]$. By now timing the motion the robot can for example cross twice before the obstacle (green path) or wait twice for it to pass (blue path). Remark that it depends on the dynamic constraints of the robot whether it is actually fast enough to cross before the obstacle.

If this method constrains the robot's motion to a single path in $\mathcal{C}_{\text{free}}$ the maneuverability of the robot is substantially decreased. Therefore, this method is typically applied to roadmaps, because they provide a set of paths in $\mathcal{C}_{\text{free}}$ that can be used deal with moving objects. Van den Berg and Overmars (2007) introduce a method that can be applied to roadmaps and deals with multiple moving objects. In the first step a collision-free roadmap with respect to static obstacles is built that encodes the kinematic constraints. Given such a roadmap an approximately time-optimal path on the roadmap is planned that obeys the dynamic constraints on the robot and avoids collisions with any of the moving obstacles. The approach is also applicable to approximate cell decompositions.

Because motion-timing does not consider the movement of obstacles directly during the search, it has a negative influence on the completeness and optimality of the planner.



(a) A straight planned path in the workspace.

(b) Two possible paths in $\mathcal{ST}$.

Figure 6.2: A planned path that is intersected by an obstacle with a known trajectory (a) can be transformed to a motion that avoids the obstacle by considering the problem in $\mathcal{ST}$ (b).

6.2.2 Direct: State $\times$ Time Approach

The search for a solution can also be performed directly in $\mathcal{ST}$. Methods that use this approach extend the procedure of direct trajectory planning by including a time dimension and mapping obstacles into $\mathcal{ST}_{\text{free}}$ as forbidden regions. The notion of time allows the planner to plan motions that are optimal in the sense of time.

The effectiveness of this approach is proved by several methods. Kushleyev and Likhachev (2009) use the state $\times$ time approach on a grid to deal with multiple dynamic obstacles in a cluttered environment. After a certain point in time the dynamic obstacles and the time dimension are discarded from the search space, to reduce the complexity the planner. This sacrifices optimality however. Phillips and Likhachev (2011) do not prune the dynamic obstacle trajectories, but they assume that inertial constraints (acceleration/deceleration) are negligible. The state $\times$ time approach is also combined with sampling-based methods, e.g., in the work of van den Berg et al. (2006) and Hsu et al. (2002).

The direct approach allows time-optimal paths in the presence of obstacles. Furthermore, it is complete. The addition of a time dimension obviously increases the computational complexity.

6.3 Robustness Against Uncertainty

Uncertainty can be present in a priori knowledge on the workspace, in sensing and in the execution of a motion. Similar to dealing with constraints and robustness against a dynamic environment, the uncertainty can be dealt with in a decoupled way or a direct way. The direct way is typically applied by representing uncertainty explicitly using the calculus of probability theory. This is a relatively new approach and is called *probabilistic robotics*. Directly taking uncertainty into account is an effective approach when the uncertainty is large, e.g., an unknown workspace or unknown localization. These cases are however outside the scope of this study. For more information on this topic the reader is referred to the work of Thrun et al. (2005).

Dealing with uncertainty in a decoupled way can still make a robot robust. Re-planning, discussed in Section 6.3.1. is an approach that deals with uncertainty in a priori information and sensing. The uncertainty in sensing can also be captured by bounded certainty regions as explained in Section 6.3.2. Finally. in Section 6.3.3 the use of feedback is discussed to deal with the uncertainty in execution.

6.3.1 Re-planning

A path that is planned before execution can become invalid during execution in a dynamic and uncertain environment. Consider the case for example, where an obstacle suddenly moves in front of the robot or when the robot encounters an obstacle on its path that it did not see before. A solution is to re-plan when the executed path becomes invalid, based on acquired sensor information during the execution. Re-planning can be done in a number of ways: complete re-planning, incremental search and anytime search.

Re-planning When a path that is planned becomes invalid, a new solution can be acquired by initiating a new search from the current configuration. Planners that use an exact representation method for $\mathcal{C}_{\text{free}}$, such as a Voronoi-based roadmap, also have to recompute $\mathcal{C}_{\text{free}}$, when it becomes invalid.

Incremental Search In an incremental search the information of previous searches is reused to find a potential solution faster in case the path becomes invalid. The focus is on applications to approximate representation methods, especially the approximate cell decomposition combined with an A* search algorithm. These methods change the A* algorithm in such a way that it can deal with costs of nodes that change during execution. The algorithms that belong to this class are known as dynamic A* or D* (Stentz, 1997) and the computationally more efficient, and recent, D* Lite (Koenig and Likhachev, 2005). An incremental search can recalculate a plan up to two order of magnitude faster than complete re-planning (Koenig and Likhachev, 2005).

Anytime Search An anytime search has an *anytime* character, i.e., it can hand over a plan at any time. This plan is approximate and suboptimal, but it is improved while time is available. So this method allows interleaving planning with execution instead of first planning and then executing. Likhachev et al. (2008) combine an incremental and anytime search and prove for a partially and completely unknown workspace that the solution found is close to optimal. Please note that Likhachev et al. (2008) regard the uncertainty as a form of a dynamic environment as opposed to this study.

In general it can be remarked that the more uncertain the environment is, the higher the need is for a planner that can recompute a plan in real-time. Re-planning algorithms suit this purpose but guaranteeing completeness proves to be difficult. Situations can occur where a robot will oscillate forever between two possible paths to the goal. A typical example occurs for robots that use a global path planner and a local motion planner (Zhang et al., 2012). The local planner might encounter a non-feasible part of the global plan (e.g., a too sharp turn or an appearing obstacle) and trigger a re-plan. When executing the new plan the non-feasible part goes out of range of the local planner. A new re-plan by the global planner can cause the robot to route down the initial plan again. Zhang et al. (2012) discuss this issue and suggest a solution that is complete. Besides being complete, re-planning methods in general are optimal, or close to optimal for anytime searches, in case of uncertainty.

Re-planning methods are also used to gain robustness against a dynamic environment. The robustness is dependent on the rate of the change of the environment. A re-plan can be efficient if the movement of obstacles is limited. But when obstacles are constantly moving, and thus re-plans are required more often, re-planning methods can fail to reach the goal or yield non-optimal plans as they do not take into account the movement of obstacles. Consider for example the motion planning problem that cyclists face when crossing a road, as illustrated in Figure 6.3. As the re-planning algorithm does not take moving obstacles into account it will yield a motion that is non-optimal in the sense of distance traveled and energy use, while a planner that does take it into account will be optimal.

It is of interest how re-planning, which computes its solution offline, relates to an online motion planner, which is able to plan a motion in real-time. The transition area between online and offline planners is vague. Offline planners that run at a high rate approximate an online planner.

6.3.2 Bounded Uncertainty Regions

Uncertainty can be captured in bounded uncertainty regions. A motion planner gains robustness by inflating obstacles with a radius that is equal to the uncertainty in the position of obstacles. The

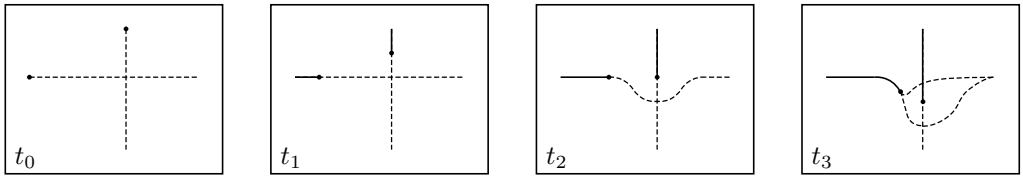


Figure 6.3: An example of a motion planning problem with two cyclist crossing, where optimality is sacrificed if the movement is not taken into account with a time dimension. At every time step the planner interleaves sensing with execution. At t_2 the motion planner decides to pass on the right. At t_3 the cyclist either has to brake and evade to the left or still tries to pass on the right. A motion planner that accounts for the movement of the crossing cyclist would have generated a path that went straight on.

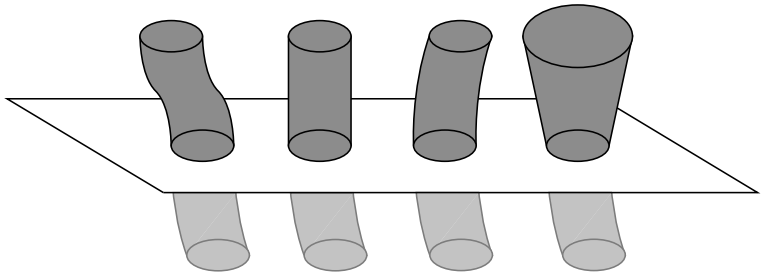


Figure 6.4: Uncertainty in obstacle trajectories can be modeled in different ways. An obstacle trajectory can be known beforehand (left), it can be assumed static (center left), extrapolated (center right) or predicted with a worst-case model (right).

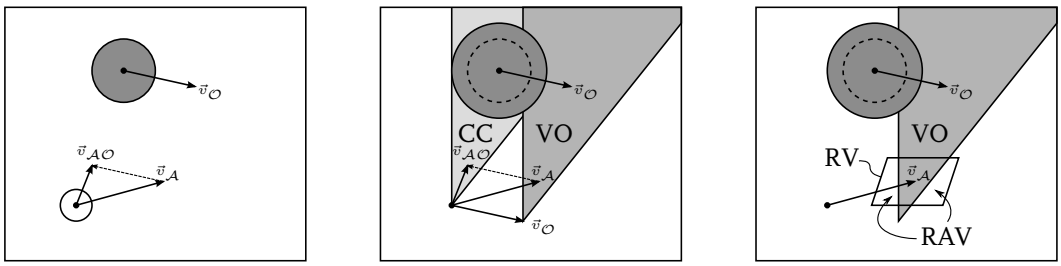
uncertainty in the execution of a motion can also be captured by adding the maximum tracking error to the inflation radius. In the case of moving obstacles the trajectories can be extrapolated based on a prediction model, as depicted in Figure 6.4. The obstacle can be assumed static or its trajectory can be extrapolated based on previous motion or a worst-case trajectory could be used based on current position and a maximum velocity. Completeness and optimality are sacrificed when obstacles are inflated. The more uncertainty, the 'larger' the obstacles become and the bigger the chance is that a path is non-optimal or that the planner is incomplete. An example of this appeared in Figure 2.4.

6.3.3 Feedback

So far it has been assumed that a continuous path solves a motion planning problem. Future configurations may not be predictable during execution. Section 6.3.2 introduced the bounded uncertainty region to capture this uncertainty. The uncertainty can also be reduced. A traditional way to account for this in robotics is to use a feedback control law that attempts to track the computed path as closely as possible. This is satisfactory, but it is important to recognize that this approach is decoupled. Feedback and dynamics are neglected in the construction of the original path; the computed path may therefore not even be usable. For example, overshoot during the execution of a trajectory might cause a collision with an obstacle.

Tracking Controller A feedback control law that uses state feedback can implicitly account for the fact that future states may be unpredictable. A strictly stable tracking controller is able to track a given trajectory upon perturbations in the environment. Typically a PD controller is used to minimize the error between the robot position and the prescribed trajectory.

Potential Field Control Besides being a guidance provider the gradient of the potential field can also be used directly as part of a feedback controller. In Section 4.4.1 such feedback methods of feeding the gradient directly to the servo loops are discussed. It is important to note that an improper coupling between the gradient field and the robot's servo loops can result in undesired behavior, such as the narrow corridor artifact (Koren and Borenstein, 1991). This is the phenomena where a robot



(a) A robot $\mathcal{A}$ on collision course with an obstacle $\mathcal{O}$.

(b) The collision cone (CC) and velocity obstacle (VO).

(c) The set of reachable avoidance velocities (RAV) to avoid the collision.

Figure 6.5: The velocity obstacle approach for a circular robot in $\mathcal{W} = \mathbb{R}^2$ (a). An obstacle can be avoided by first defining the collision cone, i.e., set of relative collision velocities (b). Next, selecting a reachable velocity (RV) that is outside of the set of collision velocities (VO) will avoid a collision (c).

shows oscillatory behavior when navigating through a narrow corridor. Masoud (2005) addressed the misunderstanding in the dual role of the gradient and proposed a damping force to deal with it.

6.4 Reactive Planners

Methods that only use local knowledge of the obstacle field to plan the trajectory are called reactive planners. Reactive planners are important in dealing with uncertainty. In the case where a global obstacle map is not available and obstacle positions are known only within the sensor radius, a reactive algorithm prevents collisions by stopping or swerving the robot when an obstacle is known to be in the trajectory. This type of approach is important in many existing practical implementations in order to deal with obstacles in uncertain and dynamic environments. However, as a reactive planner does not consider the global planning problem it is rarely used without some global planner. In other words, if only a reactive planner is used, there is no guarantee that the resulting trajectory will lead to the goal, let alone it is an optimal one. Therefore a reactive planner is typically used in combination with a global path planner.

6.4.1 First-Order Methods

Reactive planners typically rely solely on the velocity instead of the position of the robot and obstacles to prevent collisions. Therefore they are also referred to as first-order approaches. The reactive behavior is achieved by mapping the dynamic obstacles into the velocity space of the robot. This is known as the *velocity obstacle* (VO) (Fiorini and Shiller, 1998; Shiller et al., 2001), i.e., the set of velocities that would result in a collision between the robot and an obstacle moving at a given velocity. The avoidance maneuver at a specific time is computed by selecting velocities that are not in that set. The set of all avoiding velocities is reduced to the dynamically feasible velocities by considering the robot's kinodynamic constraints. An example of a velocity obstacle for a circular robot in $\mathcal{W} = \mathbb{R}^2$ is given in Figure 6.5.

6.4.2 Potential Field Methods

The analytical potential field method, introduced in Section 4.4.4, is a reactive planner. It can be extended by taking into account the velocity information on obstacles as well (Ge and Cui, 2002; Munasinghe et al., 2005; Wilschut, 2011). Reactive planners have also been inspired by the biological immune system (Luh and Liu, 2007). Also fuzzy-logic is used as a basis (Mucientes et al., 2001; Pratihar et al., 1999; Zavlangas et al., 2000). Both latter methods provide a structure that allows a direct collision avoiding response to the latest sensor information.

6.4.3 Receding Horizon Control

Receding horizon control (RHC) or model predictive control (MPC) solves a numerical optimization problem over a reduced time horizon. In this approach, an open-loop control policy is designed to control the vehicle until the end of the time horizon. Optimization over a finite horizon requires reduced computation time, however, it will not yield a globally optimal solution without using an appropriate cost-to-go function to capture the discarded part of the trajectory.

The dynamic window approach (DWA) (Fox et al., 1997) is an example of a RHC method for mobile robots. It is designed for a synchro-drive robot, i.e., a robot with three or four wheels where one motor drives the wheels and one motor determines the steering angle. The approach searches for a translational and rotational velocity directly in the space of velocities. This space is spanned by the dynamic window: the set of reachable and admissible velocities, which are determined by a certain time interval and dynamic constraints. A forward simulation over the time interval is performed for all velocities in this set to check whether the trajectory is safe, i.e., if the robot can stop without collision. The final trajectory is chosen by maximizing an objective function that for example takes into account the distance to the goal, the forward velocity and the distance to the closest obstacle. Marder-Eppstein et al. (2010) show that a mobile robot using a DWA can drive a marathon distance (42.195 km) in an office environment without collision.

6.5 Other Methods and Issues

6.5.1 Use of Heuristics

In Chapter 5 it is concluded that the use of a distance heuristic to the goal can reduce the complexity. The heuristic can be informed with more than just the distance to the goal. Many planners (Likhachev et al., 2008; Phillips and Likhachev, 2011) use a search in a lower dimension as a heuristic to inform a search in the full dimension of the problem. For example, a planner that does a search in 5D $(x, y, \dot{x}, \dot{y}, t)$ can be informed with a heuristic that does a search in only 2D (x, y). This heuristic search ignores kinodynamic constraints and moving obstacles, but can reduce the search space of the 5D search substantially. Remark that the reduce in complexity depends on whether the heuristic path is close the solution in the full state of the robot.

6.5.2 Hierarchical Planning

A hierarchical planning approach solves the motion planning problem in consecutive layers. Section 2.4 introduced the notion of a global and a local planner. This is a hierarchical approach, as the problem is decoupled in planning on a global map and a local map. Such a decoupling is typically used for large-scale spaces, i.e., spaces that extend beyond the sensory horizon of a robot.

A robot typically uses a *metric* map, i.e., based on absolute geometric positions, within its sensory horizon. The representation methods in Chapter 4 form such metric maps. If the robot has to plan a path outside its sensory horizon other types maps are also applied.

A *topological* map is a more abstract representation that describes relationships among features of the environment, without any absolute reference system. Such environmental features can be landmarks or identifiable locations such as a room of a house or an intersection of roads. A topological map has the advantage of being more compact and more stable with respect to sensor noise and to uncertainty in a priori information on the workspace. The topological map does not provide a framework to control the robot, since it does not take into account the dynamics of the robot and can not capture moving obstacles. Therefore it is generally combined with a metric map in a *hybrid* approach, e.g., in the work of Zavlangas and Tzafestas (2002) and Kuipers et al. (2004).

As robots make their way into homes, offices and other public places, there is an increasing interest to capture the human point-of-view of robot environments in a so called *semantic map*. The environment provides valuable semantic information originating from humans as designers and users. The ability to understand the semantics of space and associate semantic terms like 'kitchen' or 'corridor' with spatial locations, gives a much more intuitive idea of the position of the robot than a pure metric or topological location. This semantic information can be encoded by hand, but the robot can also reason about the semantics. This approach allows human-robot interaction and it may enable a robot to perform in a more intelligent and autonomous manner (Galindo et al., 2008; Pronobis, 2011).

Although the complexity can be reduced significantly, one must be aware of possible incompleteness. A global planner that does not take into account the full state of the robot might yield a solution is not feasible. For example, if a robot plans to move from one room to another while it does not fit through the connecting door.

6.5.3 Optimization

Suppose a motion planner that returns a feasible trajectory. In a complementary step this trajectory can be optimized. By perturbing the trajectory, while satisfying all constraints, the optimality of that trajectory can be improved.
For example, trajectories can be shortened by trying to shortcut a path. Especially trajectories that are computed by sampling-based methods, e.g., the RRT discussed in Section 4.3.2, can be shortened. The randomized character can result in undesirably long trajectories. Besides shortcutting a trajectory it can also be optimized using mathematical programming methods (LaValle, 2006). The trajectory planning problem is then treated as a numerical optimization problem, using nonlinear programming for example. Such methods define a trajectory as a function of parameters. The space of parameters is incrementally searched for a solution that has parameters with a lower cost. This is done by a gradient descent. Recent examples of optimization applied to path and motion planning are given by respectively Ryu et al. (2011) and Xu et al. (2012).

A trajectory can be improved substantially in terms of optimality. However, each perturbation of a trajectory requires a collision check and integration. Furthermore, numerical optimization methods can suffer from local minima. Therefore, the potential benefit of a more optimal trajectory has to be weighted against the extra required computation time.

6.6 Conclusions

The properties of the planning approaches with respect to requirements are summarized in Table 6.1. For each approach its effect on the completeness, optimality and computational complexity of the motion planner is indicated as either positive (+) or negative (−). A positive effect of an approach on the computational complexity means that the complexity is of the same or a lower order than the basic motion planning problem. Furthermore, it is shown with what extensions of the basic motion planning problem a specific planning approach deals. The benchmark is a motion planning problem of a mobile robot that needs to execute a motion that satisfies kinodynamic constraints in an environment with moving obstacles and uncertainty in priori information on the workspace, sensing and execution.

Planning approaches can be roughly divided into decoupled and direct ones. A decoupled approach divides the problem into parts that are solved separately, while the direct approach tries the solve the problem as a whole. Their strengths and weaknesses are orthogonal. The decoupled approach is generally chosen as each decoupled part is computationally less complex to solve than the whole problem. The inherent consequence is the negative influence on the completeness and optimality. A direct approach takes into account the full state of the robot and the environment when searching for a solution. This increases the complexity, but the solution will generally be closer to optimality and completeness. An example is given in Figure 6.3, where a decoupled approach leads to an non-optimal path. A direct approach in this case would have resulted in an optimal path, but it requires a

time dimension and thus results in an increase in complexity.
The direct approach to deal with uncertainty is not treated in this study. This approach is effective when uncertainty is large, e.g., an unknown workspace or unknown localization. These cases are however outside the scope of this study. For more information on this topic the reader is referred to the work of Thrun et al. (2005).

Ideally one wants a planning approach that deals with a dynamic environment, uncertainty and its own constraints, while being complete and optimal. Computational complexity however limits the implementation of such an approach. Reactive behavior is necessary to be robust against uncertainty and a dynamic environment. This typically requires a real-time implementation. Due to (still) limited computational resources the complexity of the motion planning problem must be reduced to solve it in real-time. This can be achieved in a number of ways.

- ▷ Using a hierarchical representation. The complexity of the overall problem can be lowered by not considering the full dimension of $\mathcal{C}$ in the entire workspace. A typical approach is to use a global planner that only plans a path. Locally, within the sensor range of the robot, a motion planner is used that does consider the full dimension of $\mathcal{C}$. A topological map, introduced in Section 6.5.2, is interesting for environments that extend far beyond the sensor range of a robot.
- ▷ By approximating the representation. By choosing an approximate representation method for $\mathcal{C}$, as discussed in Chapter 4, either the search resolution (approximate cell decomposition) or the search coverage (sampling-based methods) can be reduced.
- ▷ Using heuristics in searches. Information on the problem can be incorporated in the search to substantially reduce the complexity. In Section 5.2 the A* search is introduced that uses information on the distance to the goal. Section 6.5.1 explains how a search in a lower dimension can act as a heuristic to improve a search in a higher dimension.
- ▷ Decoupling the problem. As explained in this chapter, dealing with the extensions of the motion planning problem can be done in a decoupled way. The problem is subdivided into parts that each constitute a solution to the problem but are computationally less complex to solve than the whole problem.

Lowering the complexity is generally at the cost of sacrificing completeness and/or optimality. The question is: how important are the requirements of completeness and optimality? Both are very strict requirements. In Chapter 4 it is concluded that in order to deal with arbitrary shapes and high-order a $\mathcal{C}$ completeness must be sacrificed, using approximate representation methods. For optimality a similar conclusion can be drawn. It turns out that complexity can be significantly lowered by sacrificing optimality. In practice a solution can still be close to optimal.

The requirements depend on the robot, its environment and the task it is supposed to fulfill. It is impossible to say that any method or approach is better than the other, only that it is more appropriate. The selection of an appropriate representation method, search algorithm and a planning approach for the TURTLEs and AMIGO will be discussed in the next chapter.

Table 6.1: A comparison of planning approaches showing the effect of an approach on the completeness, optimality and computational complexity of the motion planner, and with what extensions of the basic motion planning problem a specific planning approach deals. The benchmark is a motion planning problem of a mobile robot in an environment with moving obstacles and uncertainty in priori information on the workspace, sensing and execution.

Requirements	Planning approaches								
	Path planning	Decoupled trajectory planning	Direct trajectory planning	Motion-timing	State × time	Re-planning	Bounded uncertainty region	Feedback	Reactive planner
Completeness	−	−	+	−	+	$-^a$	−	−	−
Optimality	−	−	+	−	+	$-^a$	−	−	−
Computational complexity	+	+	−	+	−	+	+	+	+
Robustness against a dynamic environment				✓	✓	✓	✓		✓
Robustness against uncertainty						✓	✓	✓	✓
Dealing with constraints		✓	✓	✓	✓			✓	✓

Denotations: + indicates a positive effect of a specific planning approach on a requirement; − indicates a negative effect of a specific planning approach on a requirement; 3 shows that a specific planning approach deals with a specific extension of the basic motion planning problem.
Superscripts: ^{a} if re-planning is only used for robustness against uncertainty it has a positive effect on completeness and optimality.

Chapter 7

Motion Planning for RoboCup

In Chapter 2 the motion planning has been introduced. This chapter specifies the problem for both the TURTLEs and AMIGO.

In Section 7.1 the robot characteristics and the environment of the robot are discussed. The current implementation is introduced in Section 7.2 and in Section 7.3 its limitations are discussed. Finally, the requirements for a new motion planner are stated in Section 7.4. The section will be concluded with the proposition for a new motion planning implementation that satisfies the requirements.

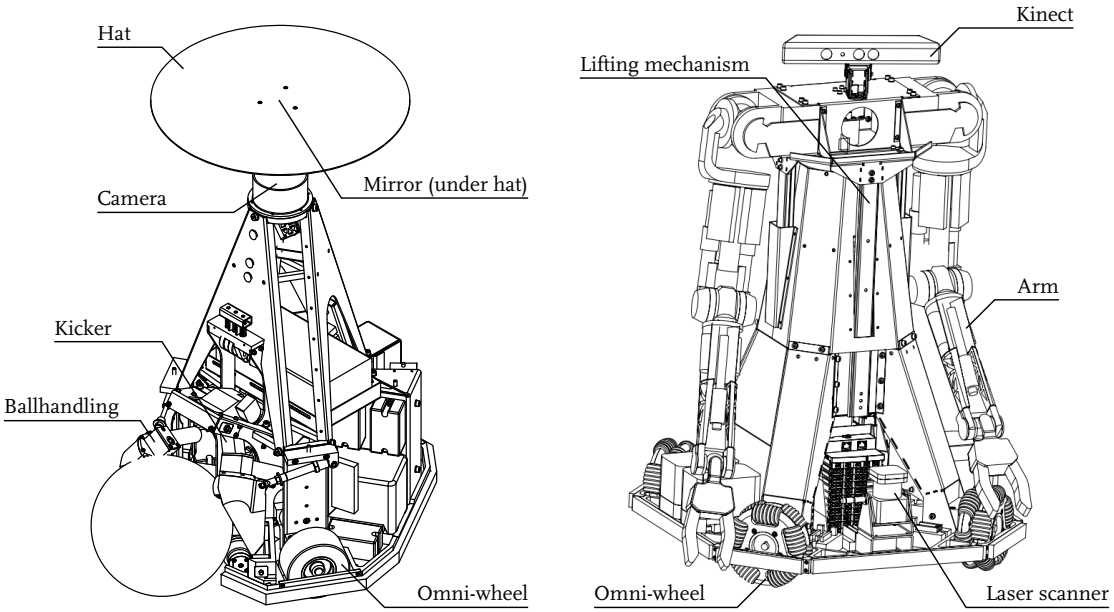
7.1 The RoboCup Environment

The TURTLEs compete in the RoboCup Soccer Middle Size League. Their environment is a soccer pitch. AMIGO competes in the RoboCup @Home League. The @Home setting is a typical household/care environment. For both robots their characteristics will be described first, followed by the characteristics of the environment. The characteristics of the robots are summarized in Table 7.1 and of their environment in Table 7.2.

7.1.1 TURTLE Versus AMIGO

The TURTLE robot, depicted in Figure 7.1a, is the first robot of team Tech United Eindhoven that started competing in the RoboCup. It is actuated by three omni-wheels. An omni-wheel is a wheel that is actuated in one direction and can roll freely in the perpendicular direction (achieved by small wheels on its circumference). This makes the platform holonomic. It uses a so-called 'ballhandling' mechanism consisting of two driven wheels to control the ball. A kicker driven by a solenoid can shoot the ball.
Each TURTLE robot relies on an omnidirectional vision system for navigation. This system consists of a full color camera that faces a spherical mirror positioned above it. The mirror reflects a 360 degrees view around the TURTLE into the camera. The view of its surroundings is usable up to six meters. The team of five TURTLEs uses a world model (de Best et al., 2010). By communicating and combining the sensor information of each TURTLE, a global view arises on all robots in the field, in particular the knowledge on both position and velocity of both team players and opponent players and the ball. This world model will enable more advanced strategy decisions and better cooperation between robots as it will provide a complete view of all relevant objects in the field in terms of position and velocity.

The design of AMIGO, illustrated in Figure 7.1b, is based on the TURTLE (Alaerds, 2010; Clephas, 2011). Similar to the TURTLE it is equipped with omni-wheels that make it a holonomic platform capable of navigating through wheelchair-accessible areas. A difference is that it uses four omni-wheels. The main reason is to prevent the robot from tipping over in the case of a sudden stop (Clephas, 2011).



(a) TURTLE. (b) AMIGO.

Figure 7.1: Both robots without their covers.

The torso is equipped with two anthropomorphic arms to perform manipulation tasks in a household environment. To extend the reach of the arms the torso is connected to the base with a lifting mechanism.
AMIGO uses a Hokuyo UTM-30LX Laser Scanner for navigation purposes. It is positioned at the base in front of AMIGO. The so-called laser range finder is used to create a 2-dimensional view of the environment. On top of AMIGO and at the bottom (not displayed in Figure 7.1b) a Microsoft Kinect camera is mounted that provides 3-dimensional information.
The fused observations from both sensors are the input of the world model. AMIGO uses this world model to store attribute information, such as position, velocity and color, about the objects it encounters, and with which it keeps track of those attributes over time (van den Dries et al., 2012).

The dimensions of both robots and their characteristics are combined in Table 7.1.

Table 7.1: The specifications of the TURTLE robot versus AMIGO.

Specifications	TURTLE	AMIGO
Height	0.8 m	1.0 - 1.4 m
Weight	40 kg	70 kg
Base		
Radius	25 cm	45 cm
Actuation	3 omni-wheels	4 omni-wheels
Typical speed of movement	3 m/s	1 m/s
Typical acceleration	3 m/s^2	1 m/s^2
Sensors		
Base	-	Hokuyo laser scanner + Microsoft Kinect
Head	Omnivision module	Microsoft Kinect

Table 7.2: The specifications of the Middle Size League environment versus the @Home League environment.

Specifications	Middle Size League	@Home League
Workspace		
Type	soccer pitch	house/care environment
Size	12 m × 18 m	arbitrary
A priori knowledge	known	partially unknown
Obstacles		
Type	dynamic	static and dynamic
Shape	polygonal	arbitrary
Typical speed of movement	3 m/s	1 m/s
Adversary	yes	no

7.1.2 Middle Size League Versus @Home League

The @Home League and Middle Size League are quite different. Generally speaking, AMIGO acts in a household or care environment that typically consists of (partially unknown) multiple rooms that are cluttered with different kinds of obstacles. Moving obstacles are assumed to move at human walking speed. This makes the environment diverse and complex. The Middle Size League environment is a soccer pitch, which is much more conditioned: the obstacles have a maximum allowable size and the pitch has a fixed dimension (defined by RoboCup competition regulations). Compared to a home environment the obstacles move at higher speed and are adversary.
The differences between both environments are evident. The soccer pitch is dynamic but conditioned. The household environment is less dynamic, as it is typically populated by fewer moving obstacles, but it contains a variety of obstacles. This imposes different requirements on the motion planners. In a home environment, for example, the representation needs to deal with arbitrary shaped obstacles in an arbitrarily large and partially unknown workspace. The representation of a soccer pitch on the other hand is more conditioned, but must allow the representation of faster moving obstacles.

The specifications of both environments are summarized in Table 7.2.

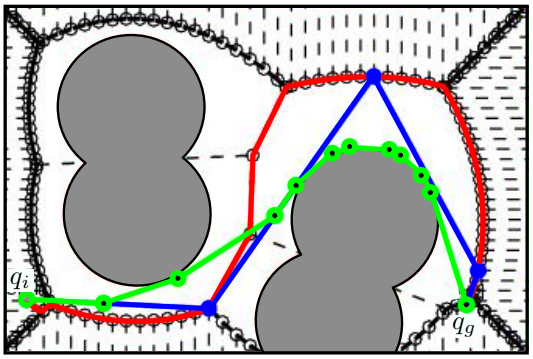
7.2 Current Motion Planners

The currently implemented motion planners for the TURTLEs and AMIGO are discussed in this section.

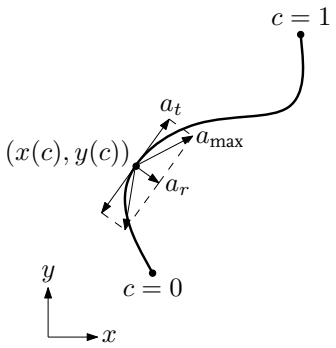
7.2.1 TURTLEs

The TURTLEs in the Middle Size League use a decoupled approach (de Best et al., 2010). It involves open- and closed-loop controllers operating at a variety of rates, linked together from top to bottom. The software is implemented with the programming language C in MATLAB (MathWorks, 2012).

The outer, open loop consists of a discrete search that produces a set of waypoints leading to the goal while avoiding obstacles. An example is represented in Figure 7.2. The workspace is represented using a *Voronoi diagram* and is searched for the shortest path with *Dijkstra's algorithm*. The second open loop level post-processes this path. A shortcut algorithm is used to cut-off sharp turns and shorten the path. This is transformed further into a shortest, collision-free path, represented by the green path in the example in Figure 7.2a. Next, the path is smoothed using Bézier curves such that the waypoints are feasible given the robot's velocity and acceleration limits, and such that the path is time-optimal. This step is shown in Figure 7.2b. Only the beginning part of the path is post-processed. The third open loop level generates a timing function $c(t)$ along the created trajectory, and creates a setpoint



(a) A Voronoi diagram for a workspace with four obstacles. The initial shortest path (red), obtained by Dijkstra's algorithm, is shortcutted (blue) and optimized in distance (green) resulting in a set of waypoints (green dots).



(b) A time optimal trajectory $c(t)$ through the waypoints, while considering the actuator limitations, is generated using Bézier curves.

Figure 7.2: The current motion planner for the TURTLEs (de Best et al., 2010). A Voronoi diagram is searched for a global path. A part of this path is post-processed to form a time optimal trajectory.

that moves through space, and finally, the inner loop is a closed-loop *tracking controller* that attempts to minimize the error between the robot and the setpoint.

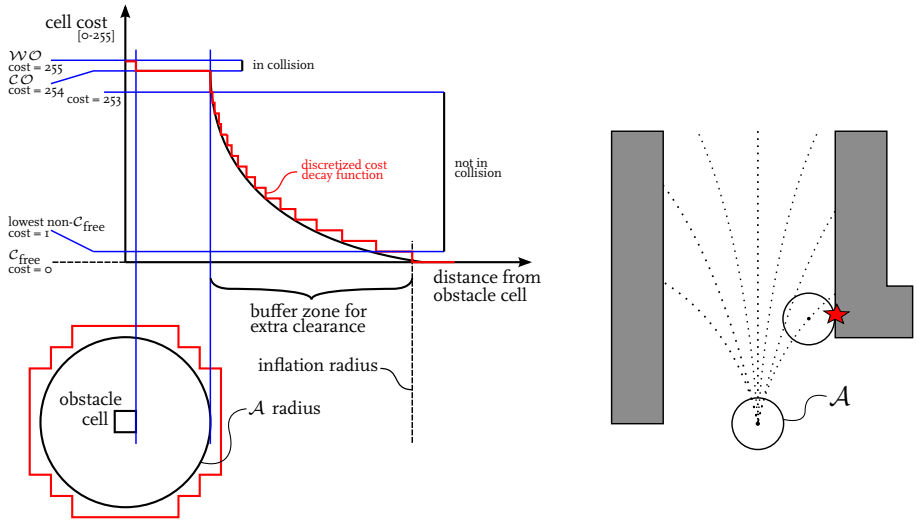
7.2.2 AMIGO

To solve a motion planning problem AMIGO also uses a decoupled planning approach (Dirkx, 2011). It uses a global path planner and a local motion planner. Both are provided by the Robot Operating System (ROS) framework (Robot Operating System (ROS), 2012), an open-source system for robot software development. The implementation is written in the programming language C++.

The global planner serves to plan a collision free path from a starting point to a goal. It computes its plan *offline*, i.e., before execution. It uses an *approximate cell decomposition*, with a grid cell resolution of 5 cm. The grid is transformed into a costmap by assigning costs to the grid cells based on the obstacles as shown in Figure 7.3a. The planner uses a *bounded certainty region*: the obstacles are inflated by a fixed radius of 30 cm to be robust against the uncertainty in sensor information and execution. A *Dijkstra's algorithm* is used to search an optimal cost path, that does not take into account the kinematics and dynamics of the robot.

The local planner computes its plan *online*. To be robust against a dynamic environment and to deal with its kinodynamic constraints it uses the Dynamic Window Approach (DWA), which is a *reactive planner*. It is seeded with the plan produced by the global planner and returns a velocity vector that attempts to follow the global plan as closely as possible. It takes the kinodynamic constraints into account as well as the obstacle information stored in the costmap. A *tracking controller* is used to minimize the error between AMIGO's velocity and the computed velocity. The global planner is queried for a new plan when the local planner is unable to compute a command velocity. The global planner then does a *complete re-plan*. During the execution the global planner update its plan typically with a frequency of 1 Hz.

While writing this study the original planner described here has been updated. Dijkstra's algorithm has been replaced by an A* search. This method returns a solution faster. The DWA approach constrains AMIGO's movement, see Section 7.3. It has been substituted for a line collision check. On a part of the global path (typically one meter ahead) a check is done whether it is still clear. If not AMIGO stops immediately and queries the global planner for a new plan.



(a) The costmap is defined by assigning the highest cost to the workspace obstacle. By adding the radius of robot $\mathcal{A}$ the $\mathcal{CO}$ is obtained. The cost function decays to $\mathcal{C}_{\text{free}}$.

(b) The DWA checks, by forward simulation (dotted lines), if a safe stop is possible for all admissible velocities.

Figure 7.3: The current motion planner implementation for AMIGO. A global search is performed on a grid with costs defined by a cost function based on obstacles. A DWA planner (see Section 6.4.3) is used locally to generate safe velocity commands.

7.3 Current Problems

Both currently implemented motion planners have their problems and limitations, which are discussed in this section.

7.3.1 TURTLEs

A number of limitations of the current planner are identified based on discussion with members of Tech United's Middle Size League team and practical experience with the TURTLEs.

- ▷ The roadmap (Voronoi diagram) as a representation method is not robust against moving obstacles. Moving obstacles can be accounted for by inflating them such that the area is covered where they could go in the time that is needed to drive from the initial to the goal position. As discussed in Section 6.3.2, this yields incompleteness. Furthermore, it can gain robustness against moving obstacles by completely re-planning at a high rate (typically 100 Hz). This will yield non-optimal motions.
- ▷ The local planner also does not take into account dynamic obstacles.
- ▷ The initial velocity of the robot is only taken into account in the local planner. This can lead to non-optimal motions as a global motion does not take into account the constraints on the robot.
- ▷ Different robot roles impose different constraints on the robot. For example, it is desirable to constrain a robot that needs to dribble the ball different than a robot without the ball. The current plan does not allow to take into account such constraints.

7.3.2 AMIGO

A number of limitations of the current planner is identified based on discussion with members of Tech United's @Home League team and tests in a hospital room environment.

- ▷ The costmap is constructed solely using the laser scan data. The Kinect must also be included to obtain a 3D costmap (using voxels) to safely navigate around obstacles that are not at the level of the laser scanner, such as a tabletop or a toy on the floor.

- ▷ In practice constantly re-planning can get AMIGO trapped in switching between two paths. For more detail the reader is referred to Section 6.3.1, where this general problem is reported. Furthermore, it can occur that the implemented Dijkstra's algorithm is unable to compute a new plan before another re-plan is requested.

- ▷ The only robustness the planner has against dynamic obstacles is acquired by re-planning. Neither the global planner, nor the local planner takes into account moving obstacles using a time dimension. This can result in an incomplete planner and non-optimal motions in the presence of moving obstacles.

- ▷ The DWA planner is designed for non-holonomic (synchro-drive) robots. This unnecessarily constrains the movement of AMIGO as it hardly utilizes its holonomic capacities. In practice this results in car-like movement especially when attaining the goal pose. It can take up to several seconds before AMIGO reaches its goal pose.

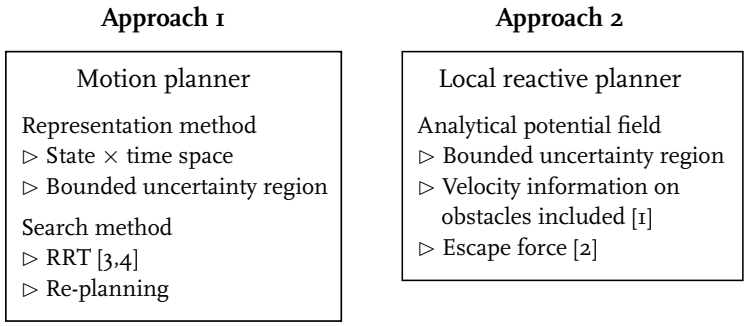
7.4 Proposed New Motion Planning Approaches

The decision for a new motion planner is based upon the requirements as formulated in Chapter 3. The requirements are dependent on the characteristics of both robots and their environment, introduced in Section 7.1. Also the current problems discussed in the previous section are taken into consideration. Based upon the interpreted requirements for both cases a number of possible implementations is presented with their advantages, disadvantages and points for further research.

7.4.1 Requirements for TURTLEs

The ultimate requirement in the soccer domain is obviously to score one more goal than the opponent. How to do this goes beyond collision-free navigation and involves a great deal of skills (e.g., dribbling, shooting) and tactics (e.g., opponent behavior, determination of a goal configuration). This study will not elaborate on such skills and tactics, but they do impose additional challenges that can be incorporated into a set of requirements for the TURTLEs.

A soccer pitch is a dynamic and adversary environment. In such an environment it is important to be ahead of opponents in order to score goals, hence time-optimal motions are desirable. This requires that obstacle trajectory must be known, but these are difficult to predict. It is therefore desirable to include the uncertainty in prediction in the search for a time-optimal solution. Furthermore, reactive behavior is necessary to be robust against opponents. This requires a real-time implementation of the motion planner. As mentioned in the conclusion on representation methods in Chapter 4 the requirement of completeness is very strict. It increases the computational complexity which is not desirable for a real-time implementation. Therefore, a resolution or probabilistic complete motion planner is accepted.
The TURTLE robot also imposes requirements on the motion planner implementation. It must satisfy its kinematic and dynamic constraints. It is important to take the velocity of the robot into account. Additionally, the dynamic interaction with the ball when dribbling could be exploited. For example, a motion planner can include the relation between ball and the robot to obtain plans to have less chance of losing the ball.



References: [1] Ge and Cui (2002), [2] Vadakkepat et al. (2000), [3] Hsu et al. (2002), [4] Zickler (2010).

Figure 7.4: The two proposed approaches for the TURTLE robot.

7.4.2 Suggested Approaches for TURTLEs

A time dimension must be included in the search for a solution in order to guarantee time-optimality. As discussed in Chapter 6 and as visible in Table 6.1, a state × time approach is an option. Guaranteeing a time-optimal solution is however difficult in the presence of moving obstacles. A second approach is using a reactive planner. It is robust against moving obstacles and uncertainty due to its reactivity, but it is not optimal.

Approach 1 A state × time approach can be implemented using an approximate cell decomposition or a sample-based method such as a PRM or a RRT. All three have a worst-case complexity that is exponential in the dimension of $\mathcal{C}$, but its complexity depends on the level of discretization or the resolution of sampling. A single-query planner like a RRT is preferred for a real-time implementation as it generally covers the reachable space the fastest, using the least nodes to obtain a solution. Furthermore, rather than attempting to discretize the continuous space and then completely explore a potentially large set of states, a RRT picks random samples directly from the underlying continuous space. This is well suited to model the continuous robot kinematics and dynamics.

Previous research on the application of a RRT on the TURTLE platform has been performed, but this approach does not consider moving obstacles and the dynamics of the TURTLE (Geerts and Naus, 2010). It must therefore be extended with a time dimension. The effectiveness of a RRT in the state × time space depends on the uncertainty in the obstacle trajectories. The closer to reality the prediction is, the more optimal the planned motion can be. On the other hand, a conservative, worst-case prediction model will exclude possible optimal motions and can even lead to incompleteness as mentioned in Section 6.3.2. An opponent velocity estimation algorithm is already available and implemented in the software. However, evaluation of this algorithm requires further research:

- ▷ The accuracy of the current estimation and prediction of obstacle trajectories needs to be determined.
- ▷ Next, it must be assessed if this accuracy is sufficient to make the state × time approach robust against moving obstacles.
- ▷ Also the limitations of the current estimation and prediction need to be defined.
- ▷ Finally, ways to improve the accuracy need to be determined.

Next, on implementation of the RRT method, the sampling strategy must be investigated. The RRT can include kinematic and dynamic constraints by sampling directly from the underlying continuous control space. This also allows to take the dynamic relation with the ball during dribbling into account. Furthermore, the sampling strategy can also be integrated with skills and tactics. E.g., bias the sampling towards regions with a higher probability of scoring a goal. Zickler (2010) adapted the RRT for these purposes and applied it in the robot soccer domain. It has been successfully tested on a team

that competes in the RoboCup Soccer Small Size League. It is interesting to implement this planning approach in the Middle Size League. To summarize, the following points are defined for future work:

- ▷ Include robot kinematics and dynamics in the sampling strategy.
- ▷ Investigate how skills and tactics can be integrated, based on the work of Zickler (2010).

Approach 2 A reactive planner is also effective in dealing with moving objects. Previous research has resulted in an analytical potential field method for the TURTLEs that has been implemented and evaluated in a MATLAB simulation environment (Wilschut, 2011). The method is reactive as it can be implemented in real-time and it also takes into account the initial velocity of the robot and the velocity of opponents. The major drawback however of the method are local minima. Wilschut (2011) solves this using an escape force as explained in Section 4.4.4. In simulation it is shown that this escape force can evade local minima created by multiple static obstacles. The question is whether this escape force is also sufficient when those obstacles are dynamic and adversary. Another practical issue is the dependence of the performance on the weighing factors of the attractive and repulsive forces. For example, a strong attractive force can be required to pull the robot through narrow passages, but in another situation a too strong attractive force might result in a collision. The use of variable weighing factors can improve the performance but this needs to be investigated in practice. Future work on the potential field method is defined as following:

- ▷ Implement the potential field method on the TURTLEs.
- ▷ Investigate the efficiency of the escape force to evade local minima and if necessary look into better alternatives.
- ▷ Investigate the use variable weighing factors to improve the performance.

7.4.3 Requirements for AMIGO

For robots in a care environment safety must be guaranteed. But how does safety relate to the requirements? Safety as a requirement can be achieved by being robust against a dynamic environment and uncertainty, and by satisfying the robot constraints, as mentioned in Section 3.1.8. Just as in the soccer domain this demands reactive behavior. Based on the latest sensor information the robot must ensure collision-free navigation. In terms of optimality a motion is required that maintains a 'safe' distance to obstacles. Besides safety also time-optimality is desired. But this can be rather conflicting as moving faster lowers the reactivity.

The computational complexity of the planner must allow a real-time implementation to be reactive. Furthermore, resolution or probabilistic completeness is required. A household or care environment is very diverse in its obstacles. Requiring completeness, requires an exact representation of these obstacles, which is computationally cumbersome. This would make a real-time implementation impossible, hence a resolution or probabilistic complete planner is necessary.

7.4.4 Suggested Approaches for AMIGO

AMIGO's environment in the RoboCup competition is a house that typically consists of three rooms. A global planner is used to navigate on a metric map of this environment. Based on this study it is suggested to further abstract this map into a topological layer. Such a representation can be used independently of the choice for an approach. The structure of rooms lends itself perfectly for a topological representation, where nodes represent rooms and edges represent the connection between rooms (e.g., through a door). The advantages of such a compact representation over a metric representation are: (1) it allows faster planning, especially when the environment is upscaled; (2) it is more stable with respect to sensor noise and to uncertainty in a priori information; and (3) it is convenient for

symbolic planners and natural language (e.g., *"go to living room"*). The topological layer can however cause non-optimality or even incompleteness, as mentioned in Section 6.5.2. The topological map can be obtained from user input or by construction (e.g., Thrun (1998)). Research is currently performed within the team of Tech United that competes in the RoboCup @Home League on probabilistic relations between objects, and objects and rooms (Jansen et al., 2012) and on an efficient representation of the world based on a specific task (Geelen et al., 2012). The former can be used to inform the topological map with semantics, while the latter can be used to represent only that part of the world that is relevant for a motion planning task. To obtain a topological map the following points must be elaborated further:

- ▷ Given a map of the workspace, distinguish rooms and recognize them (either by user input or an algorithm).
- ▷ Minimize the loss in optimality compared to the current global planner and prevent incompleteness.

A topological layer reduces the global motion planning problem to the constant problem of planning from one topological node to the other, hence it limits the complexity of the global planning problem.

For collision-free navigation of AMIGO it is strongly advised to include 3D Kinect sensor information. This can be achieved by representing a point cloud, i.e., the set of nodes produced by the Kinect, as a grid of cubic volumes (voxels) that discretize the mapped area. An efficient representation is to use an octree (Wurm et al., 2010), which is equivalent to a quad-tree in 2D (see Section 4.2.2). A 2D obstacle map can be obtained by projecting the 3D map on the ground plane as suggested by Wurm et al. (2010). Future work to include 3D information is defined as following:

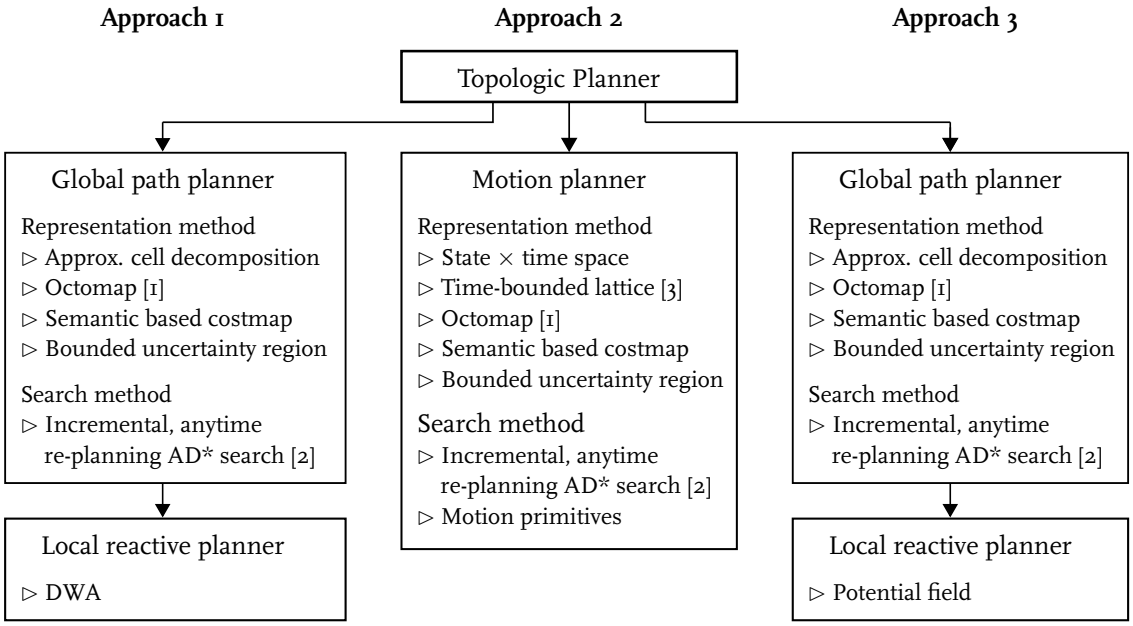
- ▷ Include 3D information from the Kinect to form a 3D map.
- ▷ Obtain a 2D obstacle map by projecting the 3D representation on the ground plane.

A second suggestion for the costmap implementation concerns the interpretation of safe behavior. In the current implementation safety measures are included in the costmap by inflating the obstacles. There is however no distinction as the obstacles are inflated with the sum of all uncertainties (the robot's tracking error, the uncertainty in the sensor information and an extra safety margin). As a result there is no understanding of safety in the costmap. This must be incorporated be defining the costmap in an intuitive, semantic way. For example, the uncertainty in the position of a human being is higher than a couch. Or maintain a certain distance from walls. This information is very valuable and should be included in the costmap. Furthermore, safety can also be incorporated in terms of velocity. As a robot slows down the constraint on its reactivity is less hard, allowing safer navigation. For example, the driver of a car can respond faster to a situation ahead when it lowers its speed. The environment can put constraints on the robot. E.g., the robot must lower its speed if it close to an uncertain object such as a door. To summarize, the costmap must be further developed on the following points:

- ▷ Represent obstacles semantically in the costmap.
- ▷ Include uncertainty information based on obstacles.
- ▷ Integrate safety constraints on the robot's position and velocity.

The topological map and the 3D costmap are representation methods. They are general and can both be used with the three following suggested approaches (in random order) to arrive a solution.

Approach 1 The first approach that is suggested resembles the current implementation. It consists of a global path planner and the DWA as a local reactive planner. The DWA planner can be real-time



References: [1] Wurm et al. (2010), [2] Likhachev et al. (2008), [3] Kushleyev and Likhachev (2009).

Figure 7.5: The three proposed approaches for AMIGO.

implemented and ensures a reactive behavior. As mentioned in Chapter 6 and as visible in Table 6.1, a reactive planner lacks in completeness and optimality. Therefore, a global path planner is suggested. Based on the research in Chapter 5 an A* algorithm is advised to arrive at a solution with the least cost. In the current implementation a DWA is already used. It needs to be redesigned however. As mentioned in Section 7.3 the current implementation hardly exploits AMIGO's holonomic capacities as it designed for (non-holonomic) synchro-drive robots. Furthermore, AMIGO can be made more robust to uncertainty by allowing faster re-planning. The downside of this approach is that dealing with moving obstacles is difficult. It is non-optimal and possibly incomplete in the presence of moving obstacles. The following number of points is identified for future work:

- ▷ Further improve the speed of re-planning using an incremental and/or anytime search, as explained in Section 6.3.1.
- ▷ Adapt the DWA such that it utilizes AMIGO's holonomic capacities.

Approach 2 A second approach is to plan in the state × time space. The advantage this approach has over the previous one is that it can take moving obstacles into account whilst maintaining optimality and completeness. (Phillips and Likhachev, 2011) prove that this is effective for mobile robots, but the downside is its computational complexity. Kushleyev and Likhachev (2009) show that relaxing the requirements of optimality and completeness, by disregarding dynamic obstacles after a fixed time period, can result in a near-optimal planner that (re-)plans fast (34 ms). In the absence of dynamic obstacles it turns from a full state planner to a kinematic planner in (x, y). Such a planning approach is considered more realistic as uncertainty in the prediction of obstacle behavior becomes too great to be of any use. The search in state × time space can be performed using motion primitives as discussed in Section 6.1.2. For the kinematic planner an anytime, incremental re-planner can be used, as introduced in Section 6.3.1. The following future work is identified:

- ▷ Represent the dynamic obstacle trajectories in the worldmodel.
- ▷ Determine if the accuracy of the trajectory estimation is sufficient to make the state × time approach robust against moving obstacles.

- ▷ Identify the limitations of the current estimation and determine ways to improve the accuracy.
- ▷ Use incremental, anytime re-planning and motion primitives in the search for a solution.

Approach 3 In Section 4.4 it is discussed that the potential field can be used as a local obstacle avoidance method and as a global navigation method. The local method allows reactive behavior, based on the latest sensor information, but is well-known to suffer from local minima that can trap the robot. Global information (outside the sensor range) is necessary to overcome this (typically using a global path planner) as discussed in Section 6.4. Especially in a complex and diverse environment such as a household a planner must rely on global information to navigate efficiently. This raises the question of what the benefit from a potential field would be then.
The benefit becomes clear if we would not only consider the base in the motion planning problem but AMIGO's full body. For example, when AMIGO needs to pick up an item from a table it also uses its arm and its lifting mechanism. This respectively adds 7 DOF's and 1 DOF to the problem! Although this problem can be solved decoupled (e.g., determine the base position first and then solve the grasping problem) it is more elegant and intuitive to solve it in one instance using full body control, as proposed by, e.g., Dietrich et al. (2012). The direction of the base will then be the result of all repulsive forces that obstacles exert on the robot and the attractive force of the goal. A potential field is well-suited for this purpose as it can define task execution in the intuitive, low-dimensional workspace (see Section 4.4.2). As this reduces the complexity of the problem it allows a real-time implementation, which is elegant for such a high-dimensional problem.
The reduce in dimension for the planning of a mobile base with 3 DOF such as AMIGO is less than for a 7 DOF arm. However, in the framework of full-body control a potential field for the base of AMIGO is an option. As mentioned this requires a global planner that avoids local minima. This can be a global path planner that is similar to approach 1. To successfully implement the potential field the following future work is defined:

- ▷ Investigate how global information can be used to guide AMIGO away from local minima.

Bibliography

Acar, E., Choset, H., Rizzi, A., Atkar, P., and Hull, D. Morse decompositions for coverage tasks. *The International Journal of Robotics Research*, 21(4):331–344, 2002.

Alaerds, R. Mechanical design of the next generation Tech United Turtle. Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2010.

Amato, N., Bayazit, O., Dale, L., Jones, C., and Vallejo, D. OBPRM: an obstacle-based PRM for 3D workspaces. *Robotics: the algorithmic perspective*, pages 155–168, 1998.

Asimov, I. *I, Robot*. Doubleday science fiction. Doubleday, 1963.

Barbehenn, M. A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices. *IEEE Transactions on Computers*, 47(2):263, 1998.

Barraquand, J., Kavraki, L., Latombe, J., Li, T.-Y., Motwani, R., and Raghavan, P. A random sampling scheme for path planning. *International Journal of Robotics Research*, 16:759–774, 1996.

Barraquand, J., Langlois, B., and Latombe, J. Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man and Cybernetics*, 22(2):224–241, 1992.

Choset, H. Coverage for robotics - A survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1):113–126, 2001.

Choset, H. and Burdick, J. Sensor based planning. I. the generalized voronoi graph. In *Proceedings International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1649–1655, 1995a.

Choset, H. and Burdick, J. Sensor based planning. II. incremental construction of the generalized voronoi graph. In *Proceedings International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1643–1648, 1995b.

Choset, H. and Burdick, J. Sensor-based exploration: The hierarchical generalized voronoi graph. *The International Journal of Robotics Research*, 19(2):96–125, 2000.

Choset, H. *Principles of robot motion: theory, algorithms, and implementation*. The MIT Press, 2005.

Clephas, T. Design and control of a service robot. Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2011.

Connolly, C. I., Burns, J. B., and Weiss, R. Path planning using Laplace's equation. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, pages 2102–2106, 1990.

Connolly, C. I. and Grupen, R. A. The applications of harmonic functions to robotics. *Journal of Robotic Systems*, 10(7):931–946, 1993.

Daily, R. and Bevly, D. M. Harmonic potential field path planning for high speed vehicles. In *American Control Conference*, pages 4609–4614, 2008.

de Best, J., Bruijnen, D., Hoogendijk, R., Janssen, R., Meessen, K., Merry, R., van de Molengraft, M., Naus, G., and Ronde, M. Tech United Eindhoven team description, 2010.

Dietrich, A., Wimbock, T., Albu-Schaffer, A., and Hirzinger, G. Reactive whole-body control: Dynamic mobile manipulation using a large number of actuated degrees of freedom. *IEEE Robotics Automation Magazine*, 19(2):20–33, 2012.

Dirkx, N. Robot modeling and navigation systems design for a service robot. Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2011.

Donald, B., Xavier, P., Canny, J., and Reif, J. Kinodynamic motion planning. *Journal of the ACM (JACM)*, 40(5):1048–1066, 1993.

Erdmann, M. and Lozano-Pérez, T. On multiple moving objects. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 1419–1424, 1986.

Feder, H. J. S. and Slotine, J. J. E. Real-time path planning using harmonic potentials in dynamic environments. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 874–881, 1997.

Fiorini, P. and Shiller, Z. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

Fox, D., Burgard, W., and Thrun, S. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4(1):23–33, 1997.

Fraichard, T. Dynamic trajectory planning with dynamic constraints: a 'state-time space' approach. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1393–1400, 1993.

Galindo, C., Fernández-Madrigal, J. A., González, J., and Saffiotti, A. Robot task planning using semantic maps. *Robotics and Autonomous Systems*, 56(11):955–966, 2008.

Ge, S. and Cui, Y. Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, 13(3):207–222, 2002.

Geelen, M., Elfring, J., Perzylo, A., and Molengraft, M. v. d. An extractable task and robot specific world representation. 2012. Submitted for publication.

Geerts, E. and Naus, G. Path planning: Rapidly-exploring random tree. `http://www.techunited.nl/wiki/index.php?title=Path_planning:_Rapidly-exploring_Random_Tree`, 2010.

Guldner, J. and Utkin, V. I. Sliding mode control for an obstacle avoidance strategy based on an harmonic potential field. In *Proceedings 32nd IEEE Conference on Decision and Control*, pages 424–429, 1993.

Hart, P., Nilsson, N., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Holleman, C. and Kavraki, L. A framework for using the workspace medial axis in PRM planners. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1408–1413, 2000.

Hsu, D. *Randomized single-query motion planning in expansive spaces*. PhD thesis, Stanford University, Stanford, USA, 2000.

Hsu, D., Kindel, R., Latombe, J., and Rock, S. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, 2002.

Jansen, S., Elfring, J., and van de Molengraft, M. Object appearance prediction and active object search using probabilistic object relations. 2012. Submitted for publication.

Kant, K. and Zucker, S. W. Toward efficient trajectory planning: The path-velocity decomposition. *The International Journal of Robotics Research*, 5(3):72–89, 1986.

Kavraki, L., Svestka, P., Latombe, J., and Overmars, M. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

Keymeulen, D. and Decuyper, J. The fluid dynamics applied to mobile robot motion: the stream field method. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–385, 1994.

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

Khosla, P. and Volpe, R. Superquadric artificial potentials for obstacle avoidance and approach. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1778–1784, 1988.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. RoboCup: The robot world cup initiative. In *Proceedings 1st International Conference on Autonomous Agents*, pages 340–347, 1997.

Koditschek, D. Exact robot navigation by means of potential functions: Some topological considerations. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 1–6, 1987.

Koditschek, D. and Rimon, E. Robot navigation functions on manifolds with boundary. *Advances in Applied Mathematics*, 11(4):412–442, 1990.

Koenig, S. and Likhachev, M. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3):354–363, 2005.

Koren, Y. and Borenstein, J. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, pages 1398–1404, 1991.

Kuipers, B., Modayil, J., Beeson, P., MacMahon, M., and Savelli, F. Local metrical and global topological maps in the hybrid spatial semantic hierarchy. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, volume 5, pages 4845–4851, 2004.

Kushleyev, A. and Likhachev, M. Time-bounded lattice for efficient planning in dynamic environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1662–1668, 2009.

Latombe, J. *Robot motion planning*. Springer, December 1990.

LaValle, S. *Planning algorithms*. Cambridge University Press, Cambridge, UK, 2006.

LaValle, S. and Kuffner Jr, J. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

Li, Z., Canny, J., and Sastry, S. On motion planning for dexterous manipulation. I. the problem formulation. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 775–780, 1989.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., and Thrun, S. Anytime search in dynamic graphs. *Artificial Intelligence*, 172(14):1613–1643, 2008.

Lozano-Pérez, T. Spatial planning: A configuration space approach. *IEEE Transactions on computers*, pages 108–120, 1983.

Lozano-Pérez, T. and Wesley, M. A. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

Luh, G. and Liu, W. Motion planning for mobile robots in dynamic environments using a potential field immune network. *Proceedings Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 221(7):1033, 2007.

Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., and Konolige, K. The office marathon: robust navigation in an indoor office environment. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 300–307, 2010.

Masoud, A. Kinodynamic motion planning. *IEEE Robotics and Automation Magazine*, 17(1):85–99, 2010.

Masoud, A. Solving the narrow corridor problem in potential field-guided autonomous robots. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, pages 2909–2914, 2005.

MathWorks. Website. `http://www.mathworks.com`, 2012.

Mucientes, M., Iglesias, R., Regueiro, C., Bugarin, A., Carinena, P., and Barro, S. Fuzzy temporal rules for mobile robot guidance in dynamic environments. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 31(3):391–398, 2001.

Munasinghe, S., Oh, C., Lee, J., and Khatib, O. Obstacle avoidance using velocity dipole field method. In *International Conference on Control, Automation, and Systems (ICCAS)*, pages 1657–1661, 2005.

Phillips, M. and Likhachev, M. SIPP: safe interval path planning for dynamic environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5628–5635, 2011.

Pivtoraiko, M., Knepper, R. A., and Kelly, A. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

Pratihar, D., Deb, K., and Ghosh, A. A genetic-fuzzy approach for mobile robot navigation among moving obstacles. *International Journal of Approximate Reasoning*, 20(2):145–172, 1999.

Pronobis, A. *Semantic mapping with mobile robots*. PhD thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, 2011.

RoboCup. Website. `http://www.robocup.org`, 2012.

Robot Operating System (ROS). Website. `http://www.ros.org`, 2012.

Rohnert, H. Shortest paths in the plane with convex polygonal obstacles. *Information Processing Letters*, 23(2):71–76, 1986.

Russell, S. and Norvig, P. *Artificial intelligence: a modern approach*. Prentice hall, 2010.

Ryu, J. C., Park, F. C., and Kim, Y. Y. Mobile robot path planning algorithm by equivalent conduction heat flow topology optimization. *Structural and Multidisciplinary Optimization*, pages 1–13, 2011.

Schwartz, J. and Sharir, M. On the piano movers' problem: II. general techniques for computing topological properties of real algebraic manifolds. *Advances in applied Mathematics*, 4(1):298–351, 1983.

Shiller, Z., Large, F., and Sekhavat, S. Motion planning in dynamic environments: obstacles moving along arbitrary trajectories. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3716–3721, 2001.

Stentz, A. Optimal and efficient path planning for partially known environments. *Intelligent Unmanned Ground Vehicles*, pages 203–220, 1997.

Tech United Eindhoven. Website. `http://www.techunited.nl/en`, 2012.

Thrun, S., Burgard, W., and Fox, D. *Probabilistic robotics*. MIT Press, 2005.

Thrun, S. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

Vadakkepat, P., Tan, K., and Ming-Liang, W. Evolutionary artificial potential fields and their application in real time robot path planning. In *Proceedings Congress on Evolutionary Computation*, volume 1, pages 256–263, 2000.

van den Berg, J., Ferguson, D., and Kuffner, J. Anytime path planning and replanning in dynamic environments. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 2366–2371, 2006.

van den Berg, J. and Overmars, M. Kinodynamic motion planning on roadmaps in dynamic environments. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4253–4258, 2007.

van den Dries, S., Elfring, J., van de Molengraft, M., and Steinbuch, M. World modeling in robotics: Probabilistic multiple hypothesis anchoring. In *Proceedings IEEE workshop on Semantic Perception and Mapping for Knowledge-enabled Service Robotics at ICRA*, 2012.

Volpe, R. and Khosla, P. Artificial potentials with elliptical isopotential contours for obstacle avoidance. In *26th IEEE Conference on Decision and Control*, volume 26, pages 180–185, 1987.

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

Wilschut, T. An obstacle avoidance algorithm for a mobile robot based upon the potential field method. Technical Report 2011-420667, Eindhoven University of Technology, Eindhoven, The Netherlands, 2011.

Wurm, K. M., Hornung, A., Bennewitz, M., Stachniss, C., and Burgard, W. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proceedings Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation ICRA*, 2010.

Xu, W., Wei, J., Dolan, J. M., Zhao, H., and Zha, H. A real-time motion planner with trajectory optimization for autonomous vehicles. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2061 – 2067, 2012.

Zavlangas, P. and Tzafestas, S. Integration of topological and metric maps for indoor mobile robot path planning and navigation. *Methods and applications of artificial intelligence*, pages 746–746, 2002.

Zavlangas, P. G., Tzafestas, S. G., and Althoefer, K. Fuzzy obstacle avoidance and navigation for omnidirectional mobile robots. In *Proceedings of the third European Symposium on Intelligent Techniques*, pages 375–382, 2000.

Zhang, H., Butzke, J., and Likhachev, M. Combining global and local planning with guarantees on completeness. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4500 – 4506, 2012.

Zickler, S. *Physics-Based Robot Motion Planning in Dynamic Multi-Body Environments*. PhD thesis, Carnegie Mellon University, Pittsburgh, USA, 2010.