Linköping Studies in Science and Technology

Thesis No. 1229

Sampling-based Path Planning for an Autonomous Helicopter

by

Per Olof Pettersson

Submitted to Linköping Institute of Technology at Linköping University in partial fulfilment of the requirements for degree of Licentiate of Engineering

Department of Computer and Information Science
Linköping universitet
SE-581 83 Linköping, Sweden

Linköping 2006

Sampling-based Path Planning for an Autonomous Helicopter

by

Per Olof Pettersson

February 2006
ISBN 91–85497–15–0
Linköping Studies in Science and Technology
Thesis No. 1229
ISSN 0280–7971
LiU–Tek–Lic–2006:10

ABSTRACT

Many of the applications that have been proposed for future small unmanned aerial vehicles (UAVs) are at low altitude in areas with many obstacles. A vital component for successful navigation in such environments is a path planner that can find collision free paths for the UAV.

Two popular path planning algorithms are the probabilistic roadmap algorithm (PRM) and the rapidly-exploring random tree algorithm (RRT). Adaptations of these algorithms to an unmanned autonomous helicopter are presented in this thesis, together with a number of extensions for handling constraints at different stages of the planning process.

The result of this work is twofold:

First, the described planners and extensions have been implemented and integrated into the software architecture of a UAV. A number of flight tests with these algorithms have been performed on a physical helicopter and the results from some of them are presented in this thesis.

Second, an empirical study has been conducted, comparing the performance of the different algorithms and extensions in this planning domain. It is shown that with the environment known in advance, the PRM algorithm generally performs better than the RRT algorithm due to its precompiled roadmaps, but that the latter is also usable as long as the environment is not too complex. The study also shows that simple geometric constraints can be added in the runtime phase of the PRM algorithm, without a big impact on performance. It is also shown that postponing the motion constraints to the runtime phase can improve the performance of the planner in some cases.

This work has been supported by the Wallenberg Foundation and COMPAS NFFP nr-539.

Department of Computer and Information Science
Linköping universitet
SE-581 83 Linköping, Sweden

Acknowledgments

The work presented in this thesis would not be possible without the help and support from several people at AIICS, the Artificial Intelligence and Integrated Computer Systems Division, at the Computer and Information Science Department at Linköping University.

First I would like to thank my supervisor Patrick Doherty who has given me the opportunity to work with such an interesting project as the WITAS-project.

Building a working UAV requires a considerable collaborative effort and the flight tests would not have been possible without all the work put in by Patrik Haslum, Per Nyblom, Tommy Persson, Björn Wingman, Gianpaolo Conte, Simone Duranti, Torsten Merz, and Erik Skarman. My thanks to all of you.

I would also like to thank David Landén, Patrik Haslum, Jonas Kvarnström, Martin Magnusson, Per Nyblom, Björn Wingman, and Mariusz Wzorek, who have given of their time to go through and comment on thesis drafts at various levels of completion.

This work is funded in part by grants from the Wallenberg Foundation and COMPAS NFFP nr-539.

Contents

Chapter 1

Introduction

The area of unmanned aerial vehicles (UAVs) is developing rapidly, and the technology has now reached a level where it is feasible to deploy UAVs for real world missions. Along with the continuing improvements of control systems that have made UAVs possible, much of the hardware has also been subject to a process of miniaturization. This opens up the possibility of small and cheap UAVs that can be used for many applications where normal aircraft are too large and expensive. Many of these future applications of smaller UAVs are at low altitude in environments with obstacles, e.g., in urban areas or even indoors. This development has led to a need for efficient algorithms for solving path planning queries in environments with obstacles.

Also for UAVs flying at higher altitude, path planning can have an important role. Military reconnaissance missions using UAVs often involve flight between a series of manually specified waypoints in order to gain intelligence from one or more interesting sites in an area. With a competent path planner available, these types of missions can be declared in a more high-level language, by specifying points of interest together with constraints on what flight paths the UAV may take, e.g., to avoid areas close to air defense or preference for flight at low altitude in order to avoid detection.

The last decade has also shown much progress in the path planning field and modern path planners have now been applied to a wide range of robots.

Much of this success comes from the introduction of good heuristic planners based on random sampling of the robot's configuration space. Two such algorithms are the probabilistic roadmap algorithm (PRM) and the rapidly-exploring random tree algorithm (RRT). This thesis describes how these algorithms can be adapted for use with an autonomous helicopter. A comparison of different algorithms is provided, both in quantitative terms of performance and in more qualitative aspects such as how a planner can be integrated into the software system of a UAV and the flexibility of the planner when used for real world applications.

1.1 Contributions

The main goal of this work has been to develop a path planner suitable for an autonomous helicopter. As the work has progressed this has lead to extensions of existing sampling-based path planning algorithms and given insights in how a path planner can be integrated in a practical robotic system. The main points described in the thesis are:

- Adaptations of sampling-based path planning algorithms for integration with the helicopter platform. This includes integration both with other parts of the high-level software architecture and with the low-level control system. Some aspects of how an autonomous helicopter system with automated path planning capabilities can be used in practice have also been considered.

- A study of how different types of constraints can be integrated at various stages throughout the planning process. This is an important issue since constraints vary in cost of evaluation and at what time point in the planning process they become known, i.e., some constraints are known in advance before the helicopter even has started while others may not be known or are left undetermined until just before the planning query is made.

- An empirical comparison of the PRM and RRT algorithms and the effects of handling constraints at different stages during the planning process. The comparison includes various measures on efficiency, e.g., planning times, path lengths and success rates.

1.2 The WITAS UAV Project

The research presented in this report has been conducted within the WITAS UAV project. WITAS (the Wallenberg Information Technology and Autonomous Systems Laboratory; pronounced *vee-tas*) was a long-term research project, which focused on the development of information technology for unmanned aerial vehicles, and its combination with low-level control and hardware platforms [14, 15]. The long-term goal of the project has been the development of a fully autonomous unmanned helicopter that can be used in applications involving photogrammetry, surveillance, monitoring of traffic and emergency services assistance. The project encompasses a variety of core functionalities and techniques such as prediction, planning, modeling scenes and events on the ground, use of those models for autonomous decisions, active vision, the design of deliberative/reactive architectures, geographical information systems (GIS), simulation tools, multi-modal ground operator interfaces to the UAV and much more. Figure 1.1 shows a picture of the Yamaha RMAX helicopter platform used in the project.

The main body of research has been pursued at AIICS, the Artificial Intelligence and Integrated Computer Systems Division, at the Department of Computer and Information Science, Linköping University. The image processing research was done in cooperation with the Vision Laboratory at the Department of Electrical Engineering, also at Linköping University. In addition, there was cooperation with a number of other different research groups both within Sweden and internationally.

1.3 Publications

Parts of this thesis have previously been published:

[44] Per Olof Pettersson and Patrick Doherty. Probabilistic roadmap based path planning for an autonomous unmanned aerial vehicle.

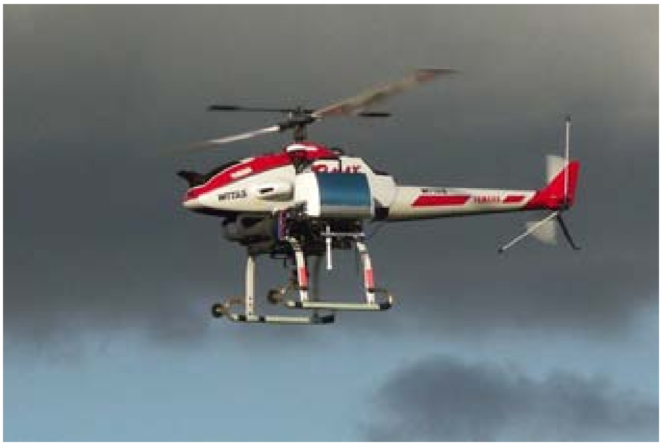


Figure 1.1: The Yamaha RMAX helicopter.

ICAPS-04 Workshop on Connecting Planning Theory with Practice, 2004. `http://www.ida.liu.se/~peope/peope-patdo-icaps04.pdf`

[45] Per Olof Pettersson and Patrick Doherty. Probabilistic roadmap based path planning for an autonomous unmanned helicopter. SAIS-SSLS 2005 Event, 2005. `http://www.ida.liu.se/~peope/SAIS05PetterssonP.pdf`

[46] Per Olof Pettersson and Patrick Doherty. Probabilistic roadmap based path planning for an autonomous unmanned helicopter. *Journal of Intelligent & Fuzzy Systems: Computational Intelligence in Northern Europe*, 2006. accepted for publication.

1.4 Outline of the thesis

The thesis has three major parts:

The first part, consisting of chapter 2–4, provides background material on path planning. The basic definition of path planning and a number of related concepts that are needed later are given in chapter 2. An overview of some early path planning algorithms is presented in chapter 3, which is followed in chapter 4 by a description of sampling-based path planning algorithms. This chapter contains descriptions of the probabilistic roadmap algorithm (PRM) and rapidly-exploring random trees (RRT) together with extensions and adaptations for different types of robotic systems.

The second part of the thesis describes the implemented path planning framework and how it has been integrated with the autonomous helicopter platform. This part begins in chapter 5 with a description of the framework and the extensions that have been made to standard algorithms. In chapter 6, the implementations of the different planning constraints are presented, most importantly the collision avoidance constraint. The integration with the helicopter and the WITAS software architecture is described in chapter 7, with some examples of how the different software components operate together during path planning scenarios.

The algorithms described in the second part of the thesis have been implemented and the results from experiments made with them can be found in the third part of the thesis. In chapter 8, some of the flight tests with the helicopter are described, and in chapter 9, a series of tests measuring the efficiency of the different planners and extensions are presented. Chapter 9 also includes discussions on the results and how they can guide the choice of path planning algorithms for different applications.

The final chapter contains the conclusions of the work presented earlier in the thesis and some topics that may be of interest for future research.

Part I

Chapter 2

The Path Planning Problem

This chapter provides definitions for basic path planning concepts, and puts them into the context of an autonomous helicopter and the planning framework developed in chapter 5. The terminology follows, with a few exceptions especially noted below, standard practice for path planning and mechanics [20, 35, 37].

In the first two sections, concepts are presented for describing the motion of a robot and the constraints that limit this motion. These definitions are used in section 2.3 for defining the path planning problem. The last section is a brief review of some complexity results from the path planning literature. It also describes a weaker completeness property called probabilistic completeness, which applies to many sampling-based path planning algorithms.

2.1 Basic Concepts

2.1.1 Work Space

The work space, $\mathcal{W}$, is the physical space in which the robot is moving. It is usually modeled as $\mathbb{R}^3$, but can also be restricted to $\mathbb{R}^2$ for robots moving in a single plane, e.g., ground robots in a one-floor building.

2.1.2 Configurations

The configuration, q, of a robot is a set of parameters that uniquely defines the location of all points of the robot in $\mathcal{W}$. For a robot with n configuration parameters, or degrees of freedom, the parameters describe a point in an n-dimensional vector space or manifold. The vector space or manifold of configurations for a robot is called the configuration space, $\mathcal{C}$, of that robot. For robots consisting of a single body that is only translating and not rotating, $\mathcal{C} = \mathcal{W}$.

The helicopter, that has served as the main test platform in the WITAS-project, is viewed as a rigid body in three-dimensional space. A full configuration of the helicopter therefore consists of a three-dimensional position vector and the following three angles describing its orientation, or attitude:

pitch the angle between the longitudinal body axis of the helicopter and the horizon.

roll the rotation around the longitudinal body axis of the helicopter.

yaw the angle between north and the direction of the helicopter body in the horizontal plane.

These angles are left unspecified by the path planner used for the helicopter. For the pitch and roll angles, this is because they are used by the low-level control system on the helicopter to achieve the requested acceleration vector for the helicopter. The yaw angle is only weakly related to the position control of the helicopter and can be chosen more freely. It was decided to let this angle also be unspecified by the path planner in order to use it for possible mission-specific needs. This can be used for pointing the helicopter in a direction suitable for certain sensors, e.g., a camera that cannot be moved freely in relation to the helicopter body.

With the orientation left undetermined, the helicopter configurations used for path planning consists of a vector in $\mathbb{R}^3$ describing the position:

$$q_{\text{heli}} = (p_x, p_y, p_z)^T \tag{2.1}$$

where p_x, p_y and p_z are the three position coordinates indicating the distance in meters from a reference point to the east, north and up respectively. Thus, this is an example where $\mathcal{C} = \mathcal{W}$.

2.1.3 State

For robotic systems in motion, not only the configuration, but also the velocity of the robot is of interest. If the configuration of a robot is described by a vector $q = (q_0, \ldots, q_n)$, the state, x, is defined as the configuration, q, together with its time-derivatives, $\dot{q}$:

$$x = \langle q, \dot{q} \rangle \tag{2.2}$$

$$\dot{q} = (\dot{q}_1, \ldots, \dot{q}_n)^T \tag{2.3}$$

If $\mathcal{C}$ is an n-dimensional vector space, the states form a $2n$-dimensional vector space, $\mathcal{X}$, that is called the state space of the robot.

The state used for describing the motion of the helicopter consists of its position together with its velocity.

2.1.4 Path Description

A path, τ, is a continuous parameterized curve in the configuration space of the robot:

$$\tau : [0, 1] \to \mathcal{C} \tag{2.4}$$

If the path is differentiable and is traversed with a speed, v, the state of the robot can be calculated as:

$$x(t) = \left\langle \tau(t), v\frac{\dot{\tau}(t)}{|\dot{\tau}(t)|} \right\rangle \tag{2.5}$$

Thus, the path fully describes the state of the robot at a point along the path except for the magnitude of the derivative, i.e., the speed of the robot.

Three-dimensional cubic C^1-splines are used in the path planning framework to describe flight paths for the helicopter, i.e., a path is described by a sequence of n cubic polynomials

$$\tau_i(s) = \mathbf{a}_{i0} + \mathbf{a}_{i1}s + \mathbf{a}_{i2}s^2 + \mathbf{a}_{i3}s^3 \qquad 1 \leq i \leq n \tag{2.6}$$

each parameterized on the interval $[0, 1]$ and continuously differentiable at knot points:

$$\tau_i(1) = \tau_{i+1}(0) \qquad 1 \leq i \leq n-1 \tag{2.7}$$

$$\dot{\tau}_i(1) = \dot{\tau}_{i+1}(0) \qquad 1 \leq i \leq n-1 \tag{2.8}$$

2.2 Constraints

The solution to the path planning problem has to satisfy various constraints that are due to properties of the robot or external factors, e.g., obstacles in the environment or given by the operator. We will differentiate between two classes of constraints: obstacle constraints that describe which configurations the robot can visit and motion constraints that describe how the robot can move through $\mathcal{C}$.

2.2.1 Obstacle Constraints

One fundamental requirement on a path planning solution is that the robot must never intersect an obstacle. The set of configurations of the robot for which no such intersection occurs is called the free configuration space, $\mathcal{C}_f$, of the robot, and path planning can be defined as finding a path that lies completely in $\mathcal{C}_f$.

One of the advantages of sampling-based path planners is that they in general do not need an explicit representation of $\mathcal{C}_f$, which can be very expensive to construct for many problems of practical interest. Instead, these algorithms probe $\mathcal{C}_f$ by testing if randomly sampled configurations are in collision with any obstacles. Thus, they only require a collision-checking algorithm that can determine if a certain configuration is collision free or not. An obstacle constraint, γ, can therefore be represented by a unary predicate on $\mathcal{C}$:

$$\gamma \subseteq \mathcal{C} \tag{2.9}$$

We also need to evaluate obstacle constraints on paths, τ, which is defined in the following manner:

$$\gamma(\tau) = \forall s \in [0, 1] : \gamma(\tau(s)) \tag{2.10}$$

Naturally, the infinite number of points along a path makes it impossible in practice to directly test for collision in this manner. This problem can be resolved either by testing a subset of the points, which is the approach taken in most of the algorithms described in chapter 4, or using a collision checker that can test complete path segments, as is described in chapter 6.

The most important obstacle constraint is the no-collision constraint that forbids the helicopter to be in contact with physical obstacles. However, for practical applications, at least in the UAV domain, it is not uncommon to also have other constraints defined in the same manner, e.g. there may be no-fly zones or limits on the altitude. These constraints can be handled together with the no-collision constraint, but due to differences in complexity of the constraints and at what time they become known to the path planner, it can be useful to treat different constraints separately. This enables the path planner to work on different sets of constraints as they become available to the planner. How this can be done is described in section 5.3.

2.2.2 Motion Constraints

Not all constraints that are relevant for path planning can be formulated as obstacle constraints. For many robots, there are further constraints on the shape of the paths that the robot can follow, e.g., a car-like robot can only travel backwards and forwards and not sideways, even if the space to the side of the robot is in $\mathcal{C}_f$.

Motion constraints, or differential constraints, differ from obstacle constraints in that they depend on derivatives of configurations. They can be classified according to the degree of the derivatives, and for motion planning the following two classes are of special interest:

Kinematic constraints are constraints where only first-order derivatives of the configuration parameters are allowed. Constraints of this type are used in motion planning to describe the valid directions of motion at different configurations, e.g., a car-like robot can only move forwards and backwards (if slipping is not considered).

Kinematic constraints on mechanical systems can be described with equations of the form:

$$g(q, \dot{q}) = 0 \tag{2.11}$$

where q is a configuration and g is a real-valued function. For motion planning, the following parameterized form is often preferred:

$$\dot{q} = f(q, u) \tag{2.12}$$

Here a control-input parameter, u, has been added, which makes it possible to simulate the system through numerical integration.

Dynamic constraints are constraints where second-order derivatives are also permissible. This class of constraints includes bounds on acceleration of the robot or its parts.

Dynamic constraints on mechanical systems can be described with equations of the form:

$$g(q, \dot{q}, \ddot{q}) = 0 \tag{2.13}$$

but also in this case, the parametric form is preferred:

$$\ddot{q} = f(\dot{q}, q, u) \tag{2.14}$$

Kinematic and dynamic constraints are examples of nonholonomic constraints. A holonomic constraint is a constraint that can be written using the following form:

$$f(q) = 0 \tag{2.15}$$

where $f : \mathcal{C} \mapsto \mathbb{R}$ [20]. If it is not possible to reduce the equation to the above form, the constraint is nonholonomic. As noted in [37], the term nonholonomic has sometimes been used in a more narrow sense in the path planning field as a synonym for kinematic constraints, especially in early work on car-like robots.

2.3 Path Planning

Motion planning is a general term for the problem of finding a plan for moving a robot from one configuration or state to another. Strict definitions of the problem vary depending on the nature of particular problems.

Motion planning without differential constraints is commonly referred to as path planning, which is defined as the problem of finding a path, τ, from an initial configuration, q_0, to a goal configuration, q_g, satisfying the obstacle constraint γ:

$$\tau(0) = q_0 \tag{2.16}$$

$$\tau(1) = q_g \tag{2.17}$$

$$\gamma(\tau) \tag{2.18}$$

This problem is also referred to as the basic motion-planning problem in [35]. It is not uncommon to also include problems with kinematic constraints in this class.

Another important class of problems is kinodynamic motion planning, i.e., motion planning under kinematic and dynamic constraints. These problems are often described in terms of a parametric state-transition function, and the problem involves finding a control input signal, u, parameterized over time, such that the system reaches the goal-state if starting at the initial state. With this approach, the solution describes the evolution of the system over time as a time-parameterized path through the state space of the system.

In this thesis, we will focus on finding a geometric path through the configuration space, but unlike the basic path planning problem defined above, we will also consider dynamic constraints in order to create paths that are well suited for the helicopter. With this approach, the path planning process and the plan execution are more loosely coupled, and the flight along the path can be performed at different speeds, and to some extent different attitudes of the helicopter.

2.4 Completeness, Optimality and Complexity

Planning an optimal path in an environment with obstacles is an intractable problem in all but the simplest cases. Even a seemingly simple problem, e.g., finding the optimal path for a point-like robot in three-dimensional space with polyhedral obstacles, is NP-hard [10]. If we want to avoid sharp corners by limiting the maximum curvature of the path, the path planning problem is NP-hard in the number of obstacle vertices, already in two dimensions [47]. These problems correspond to planning a path for a free flying helicopter with piecewise linear paths and planning for a car-like robot with limited turning capabilities.

However, by relaxing the requirements on completeness and optimality, it is possible to develop path planning algorithms that give satisfactory solutions to many problem instances. Two examples of such algorithms that will be presented in chapter 4 are based on probabilistic roadmaps and rapidly exploring random trees. These algorithms have a weaker completeness property called probabilistic completeness [24, 34]. An algorithm is said to be probabilistically complete if the probability of finding a solution converges to one, given a sufficient running time. However, these properties are mainly of theoretical interest, since the high complexity of the path planning problem makes complete planners unfeasible for most practical problem instances.

Chapter 3

Early Path Planning Algorithms

In this and the next chapter, an overview is given of the major approaches for solving path planning problems. We will begin in this chapter with a short survey of early path planning techniques before we continue with modern sampling-based planners in the next chapter. This chapter is largely based on Latombe's book on motion planning [35], but references are also given to some of the original work.

Latombe differentiates between four major approaches to motion planning: roadmaps, exact cell decompositions, approximate cell decompositions, and potential fields. Classical roadmap and cell decomposition planners are both deterministic and complete, unlike the sampling-based planners described in the next chapter. In practice, they are only applicable to problems of low dimensions, which is not surprising given the high complexity of the path planning problem.

Potential field planners differ from the above planners in that they are heuristic planners that use artificial potentials to guide a gradient descent search through $\mathcal{C}_f$. Although they are incomplete, they have been a popular choice since they are efficient for many practical problems, also in high-dimensional configuration spaces.

In this chapter, we will first go through classical roadmap and cell decomposition algorithms before we continue with potential field planners.

3.1 Classical Roadmap Algorithms

Roadmap algorithms generate a graph, called a roadmap, that represents the connectivity of the free configuration space of the robot. The roadmap is constructed in a way that makes it easy to connect the start and goal configurations to it, and when this has been done, graph search algorithms can be used for finding a path. There exist several different methods for constructing the roadmap, some of which are described in this section.

The visibility-graph algorithm is mainly applicable to two-dimensional models with polygonal obstacles. For such an environment, the roadmap can be constructed from the obstacle vertices, which are used as nodes in the roadmap. Edges are added between nodes that can see each other, i.e., two nodes are connected if there is a straight-line path between them that does not intersect any obstacle. If the graph is searched with an optimal graph-search algorithm, e.g., A*, the solution will be the shortest path possible in the model.

The visibility-graph algorithm extends also to three dimensions with polyhedral obstacles. However, the solution will no longer be optimal, since the optimal path may grace edges of the obstacles rather than corners in this case, e.g, the optimal path between the centers of two faces of a cube does not pass any corner of the cube.

Another basic roadmap construction algorithm is the retraction algorithm. Unlike the visibility-graph method, which produces paths in contact with obstacles, the retraction method finds paths that maximize the clearance to obstacles. With the retraction method, the Voronoi diagram derived from the obstacle vertices and edges is used as roadmap. A Voronoi diagram consists of points that have more than one nearest point in the obstacle region. For polygonal obstacles the Voronoi diagram is made up of straight lines between pairs of vertices or edges and of parabolic arcs between pairs with one vertex and one edge. The Voronoi diagram can be constructed in $O(n \log n)$ time [33].

The visibility-graph and retraction algorithms are mainly of interest for path planning in two dimensions, even if there are some extensions for higher dimensional configuration spaces.

A more general algorithm is due to Canny [11]. Canny's roadmap algorithm is applicable to path planning problems of any dimension, and for any type of obstacle that can be described as semi-algebraic sets, i.e., sets whose boundaries can be described with polynomial relations.

For an n-dimensional problem, Canny's roadmap algorithm uses a hyper plane of dimension $n-1$ that sweeps through the configuration space along a coordinate-axis, e_i. At each point along this axis, the boundary of $\mathcal{C}_f$ gives rise to an $n-1$-dimensional silhouette within the hyper plane. The algorithm keeps track of the extreme point of this silhouette along a different coordinate-axis, e_j, $i \neq j$. As the plane is moving, these extreme points trace curves that become edges in the roadmap. At critical points, where the set of extreme points changes, the algorithm is called recursively in the $(n-1)$-dimensional hyper plane in order to connect the different sub-graphs that are constructed from the curves traced by the extreme points.

3.2 Cell Decomposition

Cell decomposition methods are path planning algorithms based on the idea of dividing $\mathcal{C}_f$ into smaller regions in a way that makes it easy to find paths between any two points within each cell. Cell decomposition methods can be further divided into exact and approximate methods, where the former creates a partitioning of $\mathcal{C}_f$, while the latter only give approximate representations of $\mathcal{C}_f$, using cells of regular shapes.

In two dimensions with polygonal obstacles, a simple and efficient method for exact cell decomposition is vertical decomposition. This algorithm constructs the cells and a cell connectivity graph, by moving a sweep line along one coordinate axis and keeping track of the cells and edges currently intersected by that line. The cells are triangular or trapezoidal and bounded by two obstacle edges and one or two edges parallel with the sweep line. When a new obstacle vertex is reached, the current set of cells is updated by removing any cell touching that vertex. One, two, or no new cells are added in front of the sweep line, depending on the orientation of the edges extending from the vertex. These new cells are also added to the connectivity graph and connected to the cells that were removed. With suitable representations for active cells and edges during the sweep, this construction can be performed in $O(n \log n)$ time.

When the cell decomposition has been made, path planning queries can be resolved by locating the cells containing the start and goal points and performing a graph search in the connectivity graph.

Approximate cell decomposition differs from exact cell decomposition in that it divides the configuration space into regular regions of some shape, normally rectangles. These regions are labeled as either empty, if they are collision free, full, if they lie completely inside obstacles, or mixed otherwise. The construction of the cells is done recursively, and a tree of cells is built. In two dimensions each rectangle is divided along both dimensions which gives four sub-rectangles, and the tree will be a quad tree. In three dimensions a similar construction gives rise to an oct tree.

To find a solution to a path planning query, the cells containing the start and goal points are first located. The connectivity graph of these cells are then searched for a path containing only empty or mixed cells. If there are mixed cells in the path, these are decomposed further, and a new graph search is performed. This process is repeated until a path only consisting of empty cells is found.

3.3 Potential fields

The use of potential fields for path planning was introduced by Khatib, who described such an algorithm for use with a robotic arm [32].

Potential field planners use a potential defined over $\mathcal{C}_f$, which is used for guiding the robot towards the goal, while avoiding obstacles. The potential field is calculated as the sum of an attractive potential that guides the robot towards the goal and a repulsive potential that steers it away from obstacles. For the attractive potential, the square of the distance to the goal can be used. The repulsive potential should tend to infinity when the robot approaches an obstacle, but be zero when the robot is far from any obstacle.

To plan a path for a robot, a gradient descent search is performed, by taking the gradient of the potential field and applying that as a force on the robot. This process can be performed in advance for a known environment by simulating the robot's motion, or it can be performed online. Online planning can also be done in unknown environments since the potential field method only uses local information about obstacles.

The use of only local information to determine the direction of motion has a disadvantage in that the robot can get stuck in local minima. Several different solutions to this problem have been devised (some of which are described in Latombe's book [35]). One of the most significant is the Randomized Path Planner (RPP) by Barraquand and Latombe [3]. This planner alternates between greedy search on a grid in the potential field and random walks. The greedy search is performed until the algorithm detects that it has reached a local minimum with all neighbors having a higher potential. It then switches to random walk for a number of iterations, before again performing a greedy search. This process is repeated until the goal is reached.

Chapter 4

Sampling-based Path Planning Algorithms

Sampling-based path planners have been successfully applied to a wide variety of robotic systems and represent today the standard method for solving path planning problems. They differ from classical algorithms in that they do not process an explicit geometric description of $\mathcal{C}_f$. Instead they sample robot configurations randomly from $\mathcal{C}_f$ and attempt to connect these by means of a robot-specific local path planner. Since the algorithms can be adapted to many different robots by simply plugging in a suitable local planner, they are much more generic than most classical path planning algorithms. This path planning approach also provides a better separation between the planning algorithm and the representation of the environment since the only interaction with the model of C_f is through testing configurations and paths for collisions.

There are both multiple-query and single-query variants of sampling-based path planning algorithms. Planners of the first type use precompiled data structures to quickly solve multiple queries in the same environment, while planners of the second type are optimized to directly solve a single query. The most popular algorithms from these two categories are the probabilistic roadmap algorithm (PRM) and the rapidly-exploring random tree algorithm (RRT) respectively. These two algorithms will first be described for holonomic robots, together with some extensions and theoretical results. After that, methods for handling kinematic and dynamic constraints are presented. The last section is devoted to extensions for handling time and change in path planning.

4.1 Probabilistic Roadmaps

The introduction of the probabilistic roadmap algorithm (PRM) [29] represents an important milestone in the path planning field. It made it possible to solve challenging problems for robots in high dimensional configuration spaces that were far beyond the capabilities of earlier path planning algorithms.

In this section, the basic PRM algorithm will be presented first, after which some of the numerous extensions for improving roadmap coverage and the efficiency of the algorithm are described. In the final subsection some theoretical results are presented.

4.1.1 The Probabilistic Roadmap Algorithm

The PRM algorithm is a two-stage algorithm as illustrated in figure 4.1. First, a graph is constructed representing the free configuration space, $\mathcal{C}_f$, of the robot. This graph is then used for quickly solving path planning queries with conventional graph-search algorithms.

The roadmap construction algorithm, shown in algorithm 4.1.1, starts by picking n random configurations in $\mathcal{C}_f$ that become the nodes of the graph. An example is shown in figure 4.2(a) where a number of random points have been placed around two obstacles. The next step is to connect the configurations with a local path planner, P_L, suitable for the particular robot. The local path planner is a simple deterministic planner that produces paths that match the motion capabilities of the robot but ignores the obstacles. In figure 4.2(b), the points are connected with a local planner that produces straight lines. The paths generated by the local planner are checked for collisions, and only the paths that are collision free (continuous lines in the figure) are added as edges to the graph.

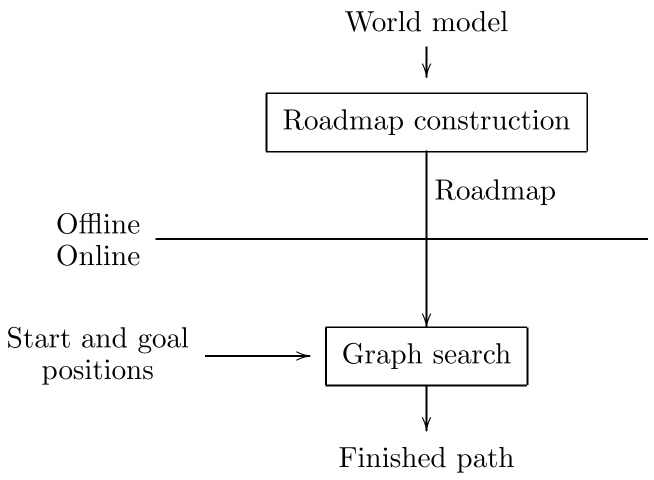
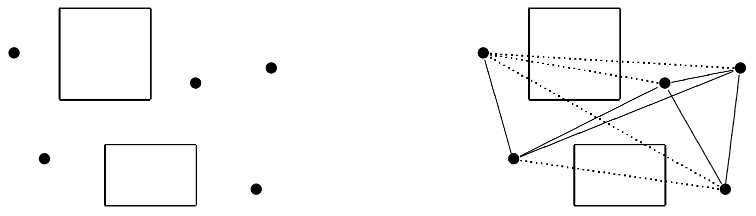


Figure 4.1: The stages of PRM planning, with an offline roadmap construction phase and the online graph-search.



(a) Adding five random points.

(b) Connect nodes if possible (continuous lines), but not nodes where a connection would intersect an obstacle (dotted lines).

Figure 4.2: Construction of the roadmap.

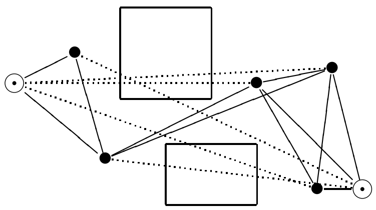
Algorithm 4.1.1 The algorithm for constructing a roadmap in the configuration space, $\mathcal{C}$, of the robot, under the no-collision constraint, γ_c, using the local path planner, P_L.

```
ConnectNode($q, \gamma_c, P_L, \langle N, E\rangle$)
 1  $N_q \leftarrow \{n \mid n \in N, d(q,n) < r\}$ /* Nodes within distance $r$ */
 2  for $n \in N_q$ in order of increasing $d(q,n)$ do
 3     $\tau \leftarrow P_L(q,n)$                      /* Local Path */
 4     if $\gamma_c(\tau)$ then
 5        $E \leftarrow E \cup \{\langle q,n\rangle\}$
 6     end if
 7  end for

MakeRoadmap($\mathcal{C}, \gamma_c, P_L$)
  1  $N \leftarrow \emptyset$           /* Set of Nodes */
  2  $E \leftarrow \emptyset$           /* Set of Edges */
  3  for $i = 0$ to $n$ do
  4     repeat
  5        pick $q$ randomly from $\mathcal{C}$
  6     until $\gamma_c(q)$
  7     $N \leftarrow N \cup \{q\}$
  8  end for
  9  for $q \in N$ do
 10     ConnectNode($q, \gamma_c, P_L, \langle N, E\rangle$)
 11  end for
```

Algorithm 4.1.2 Plans a path between q_0 and q_g, satisfying the no-collision constraint, γ_c, by using the roadmap graph, G, and the local path planner, P_L.

```
PRM QUERY($q_0, q_g, \gamma_c, P_L, G$)
1  CONNECTNODE($q_0, \gamma_c, P_L, G$)
2  CONNECTNODE($q_g, \gamma_c, P_L, G$)
3  **return** A*-search ($q_0, q_g, G$)
```


(a) To solve the planning problem, the start and goal points are added to the graph.

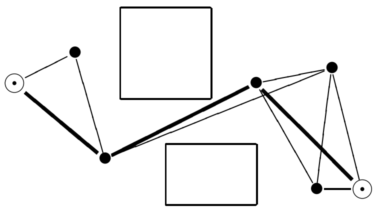

(b) The resulting graph can then be used for solving the planning problem with standard graph-search algorithms.

Figure 4.3: Online planning using the precompiled roadmap.

The second stage of the algorithm is performed online, when the robot needs to move between two configurations, and is shown in algorithm 4.1.2. The start and goal configurations are added to the graph in the same manner as when the graph was built, by trying to connect them to nodes already in the graph using the local path planner (figure 4.3(a)). If this succeeds, the planning problem can be solved with standard graph search algorithm, e.g. A*, as is shown in figure 4.3(b), and the planner returns a series of waypoints in $\mathcal{C}_f$. The waypoints specifies a unique path that can be retrieved by connecting them with the local path planner. Since the local path planner is deterministic, the same path segments are generated as during the roadmap construction phase. If the environment hasn't changed, the path generated in this manner is guaranteed to be collision free since each segment was checked during the roadmap construction phase.

If a connection attempt or the graph search fails, the planner reports that no plan was found. For difficult environments, more elaborate schemes for connecting the nodes can be used. A random walk or random bounce-walk[1] can be initiated from the start- or goal-configuration in hope of finding a path that leaves the problematic area where the configuration is located [29, 41, 42].

After a path has been found, it is usually subjected to a smoothing procedure, to eliminate the unnecessary jaggedness of the path that often occurs as a result of the probabilistic nature of the algorithm.

4.1.2 Improved Roadmap Construction

Due to the uniform sampling scheme of the standard PRM algorithm, the same number of nodes and edges are tried in open regions in C_f as in more complex regions. Since the chance of finding collision free nodes and edges is smaller in these more complicated regions, fewer nodes and edges are added there, where they are needed the most. In order to bias the roadmap construction towards more difficult regions and thereby produce better roadmaps, with fewer nodes, many different roadmap construction schemes have been devised. Some of the more important ideas will be described in the following subsections.

Sampling near obstacles

Difficult regions generally lie close to obstacles, and from this observation it is natural to seek sampling schemes that place nodes in close proximity of obstacles.

In [42], where a planner is described for car-like robots in a two-dimensional polygonal environment, an extra node is added outside each edge and vertex of the obstacles. At these nodes, the robot is oriented

[1]Random bounce-walk is a variant of random walk where the walk starts in a random direction and then changes direction upon hitting obstacles until a certain distance has been traveled or a maximum number of iterations have been performed.

perpendicular to the normal of the obstacle vertex or edge. A number of tests shows that this form of geometric node adding gives a large boost in efficiency for environments with rooms and corridors, as long as not too many narrow passages are present.

However, relying on an explicit representation of $\mathcal{C}_f$ is generally avoided, since such a representation can be very expensive to construct. In [2], this is accomplished by iterating over the obstacles and placing nodes at the intersections between the obstacle surface and rays extending from a central point of the obstacle. The distance to the surface is found with a binary search, by testing at what distances the point is in collision with the obstacle.

Another way of focusing the node distribution to the obstacle boundaries is called Gaussian Sampling [8]. This method is inspired by the image processing operation Gaussian blur, and the idea is to use the blurred image of the obstacles as a sample distribution for a PRM planner. The Gaussian blur distribution is achieved implicitly, without any processing of the configuration space besides collision checking, by the following sampling method: Nodes are sampled pair-wise, by first picking a random configuration in $\mathcal{C}$ and then picking a nearby configuration from a Gaussian distribution centered at the first configuration. If one of the two configurations is collision free while the other is not, the collision free configuration is added to the roadmap. Due to the requirement that the robot intersects an obstacle in one of the configurations, no nodes will be placed far from obstacles.

This idea was taken one step further in [22], where it was observed that even if sampling close to obstacles does indeed improve the ability of the PRM planner to find narrow passages, it is often the case that large parts of the boundary surfaces of the obstacles are uninteresting for finding solutions to path planning problems. Instead, a sampling scheme is proposed where short line segments are randomly generated. If the two end points intersect with obstacles, but the middle point is collision free, the middle point is added to the roadmap. The idea behind this sampling scheme is that when the middle point is collision free and the two end points are not, it is likely that the middle point is between two obstacles. Since the line segments span over the free configuration space from two points within obstacles, this method is called the bridge test. Some nodes are also required in more open areas of the configuration space, so some additional configurations are sampled uniformly, just as with the standard algorithm.

Another method for finding paths through narrow passages is presented in [23]. The idea here is to first build a roadmap for a dilated free space, where some degree of penetration of the robot into obstacles is allowed. This has the effect of widening narrow passages, which increases the chance of connecting different parts of the free space. To arrive at a roadmap that correctly represents the free space, the nodes and edges that intersect obstacles are pushed into the true free space of the robot. For nodes, this is done by testing a number of random configurations in a circular ring around the node and moving the node to the location of a collision free node if any such node is found. After the nodes have been relocated, each edge is tested to see if it is collision free. For edges that are not collision free, an attempt to reconnect the end-points is done by building a small roadmap with nodes sampled in a local, rectangular region between the two nodes.

Focus on Difficult Areas

Focusing the sampling distribution on difficult areas can also be done by using information gained through the roadmap construction itself. The roadmap is generated to help find plans, and as a side effect of its construction, it also provides information on which parts of the configuration space that are difficult with respect to path planning.

In [29, 30], the learning phase is divided into a number of stages. First, nodes are sampled with a uniform random distribution as in the basic PRM algorithm. In the second stage, the sampling is instead focused on improving the connectivity of the roadmap by placing samples in difficult areas. This is done by assigning a weight to each node from the first stage that describes how difficult the area is around the node. In [29], this weight is calculated as the failure ratio for connection attempts involving the node in question, while in [30] it is the inverse of the degree of the node. Other alternatives are also presented in these papers. The sampling in the second stage is performed by selecting a node in the roadmap randomly, with a probability proportional to the weight of the node, and then picking a new random configuration in its neighborhood. The expansion is made by performing a random bounce walk from the new configuration. After a certain number of iterations of random walk, connection attempts are made from the new configuration to the roadmap as in the first stage.

To further increase the chance of connecting separate components, an additional, third stage is added in [30] where several attempts are made to connect nearby nodes from each pair of components by calling a single-query-planner.

Visibility-based PRMs

The algorithms in the two previous sections focus the search to difficult areas by biasing the sampling to such areas. In the visibility-based PRM algorithm [36, 50, 51], a good roadmap is instead sought for by avoiding to add redundant nodes in regions already covered by other nodes. To accomplish this, two types of nodes in the graph are distinguished: guards and connections. Guards are nodes that are intended to cover as much as possible of C_f, while connection nodes, as the name suggests, connect the guards together. A central idea used in this algorithm is the visibility of a guard, which is the part of C_f that is reachable from the guard using the local path planner.

As with the basic algorithm, the visibility-based roadmap is constructed by repeatedly picking random configurations for the robot. For each new configuration, a check is made to see if any guards already present can be reached from the new configuration by using the local planner. There are three possible cases depending on how many guards that are seen:

i) No guard is reachable from the new configuration. The configuration is added to the set of guards.

ii) At least two guards that are not yet graph connected are reachable from the new configuration. In this case, the newly generated configuration is connected to the guards and added to the set of connections.

iii) Only one guard, or guards belonging to a single connected component, can be reached. Since this configuration already lies in the visibility domain of some guard and it does not improve the connectivity of the graph, it is discarded.

This sampling scheme gives a roadmap where two guards will never see each other directly, but can be connected to each other by connection nodes located in their common visibility area.

4.1.3 Lazy PRM

In most practical applications of the PRM algorithm, the most computationally expensive operation is the collision checking of configurations and paths. For single-query applications, only a small subset of the edges is normally visited during graph search, which makes many of the calls to the collision checker unnecessary. This observation has led to a variant of the probabilistic roadmap algorithm called lazy probabilistic roadmaps, where the collision checking is postponed until the query phase [6, 7].

During the roadmap construction phase of the lazy variant, edges are added between all pairs of nodes, or pairs of nodes that are within a certain distance from each other, without considering collisions. A planning-query is resolved by a standard graph search, and the segments of the resulting path are checked for collision. If all segments of the path are collision free, the path is a solution, otherwise the offending segments are removed from the roadmap, and a new graph search is made. These operations are done repeatedly until a collision free path is found, or the graph search fails, which means that no solution could be found. The advantage of this method is that only a subset of the edges in the roadmap has to be checked for collision, which reduces the combined time for roadmap construction and graph search.

4.1.4 Theoretical Results

The probabilistic roadmap algorithm proved already from the start to be a very successful algorithm for practical applications. Efforts have also been made to investigate the theoretical aspects of the algorithm, and how its performance depends on properties of the free configuration space, $\mathcal{C}_f$.

Two such properties for describing $\mathcal{C}_f$ are ϵ-goodness [31] and $(\epsilon, \alpha, \beta)$-expansiveness [25]. ϵ-goodness describes the degree of isolation of a point in $\mathcal{C}_f$, by setting a lower limit on the visibility region, $\mathcal{V}$, of the points in $\mathcal{C}_f$. A point, $p \in \mathcal{C}_f$, is said to be ϵ-good if

$$\mu(\mathcal{V}(p)) \geq \epsilon\mu(\mathcal{C}_f) \tag{4.1}$$

where μ is a volume measure for $\mathcal{C}$. This formula means that all points in an ϵ-good configuration space must be reachable from at least an ϵ fraction of $\mathcal{C}_f$. The free space, $\mathcal{C}_f$, is said to be ϵ-good if all points, $p \in \mathcal{C}_f$, are ϵ-good.

A first step towards a proof of probabilistic completeness for the PRM algorithm was taken in [31]. Here it was shown that given a sufficient number of nodes in the graph, the start and goal configuration can be added to the graph with a high probability. The minimum number of nodes required to achieve a connection probability of $1 - \beta$, was shown to be

$$n = \frac{c}{\epsilon}\left(\ln\frac{1}{\epsilon} + \ln\frac{4}{\beta}\right) \tag{4.2}$$

where c is a constant large enough so that $(1-x)^{(c/x)(\ln 1/x + \ln 4/\beta)} \leq x\beta/f$.

Being able to connect the start and goal configurations to the roadmap is not sufficient to guarantee a solution. The roadmap must also correctly describe the connectivity of the free space, i.e., if it is possible to find a path between two nodes, they must be in the same roadmap component. In [25], the notion of $(\epsilon, \alpha, \beta)$-expansiveness was introduced for describing the degree of connectivity that exists between subsets of connected components of $\mathcal{C}_f$. For a subset, S, of a connected component, $F \subseteq \mathcal{C}_f$, the lookout set is defined as

$$\text{lookout}_\beta(S) = \{q \in S \mid \mu(\mathcal{V}(q) - S) \geq \beta\mu(F - S)\} \tag{4.3}$$

where $\beta \in (0, 1]$, i.e., the lookout set is the part of a subset that can see a reasonably large part of the rest of the connected component. An $(\epsilon, \alpha, \beta)$-expansive free configuration space is an ϵ-good free configuration space where all connected subsets, S, of a connected component in $\mathcal{C}_f$, has a lookout-set that sees a sufficient portion of the rest of the component, i.e.,

$$\mu(\text{lookout}_\beta(S)) \geq \alpha\mu(S) \tag{4.4}$$

For free spaces that are $(\epsilon, \alpha, \beta)$-expansive it is shown in [25] that with $2\lceil 8\ln(8/\epsilon\alpha\gamma)/\epsilon\alpha + 3/\beta\rceil + 2$ nodes in the roadmap, the probability is at least $1-\gamma$ that there is only one roadmap component for each connected component of the free space.

This result, together with the result on the probability of connecting the start and goal configuration to the roadmap, proves that the probabilistic roadmap algorithm is probabilistically complete.

For practical applications, the above results are hard to apply directly, since the parameters are hard to calculate in any but the simplest free spaces. In [28], an alternative analysis is done, where the minimal path-clearance is instead used as a parameter for describing the complexity of the free-space from a planning perspective. The proof is built on the observation that if there is a path of length L between two configurations with clearance R, it is possible to place $2L/R$ balls along the path so that any point in two neighboring balls can be connected with a collision free straight-line path. If N nodes are sampled uniformly for the roadmap, the probability of not having any of them fall within a specific ball is

$$\left(1 - \frac{\pi R^d}{2^d}\right)^N \tag{4.5}$$

where d is the dimension of $\mathcal{C}$. Given that there are $2R/L$ balls along the path, the probability that there is a ball without a node is less than

$$\frac{2L}{R}\left(1 - \frac{\pi R^d}{2^d}\right)^N \tag{4.6}$$

which is also an upper bound of the failure probability for the PRM algorithm.

A collection of all the above results, by the same authors, can be found in [24].

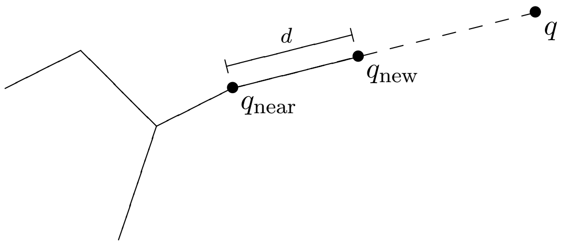


Figure 4.4: The expansion step of the RRT algorithm. A configuration, q, is picked randomly from $\mathcal{C}_f$ and an extension of length d is made towards it from the closest node already in the tree, q_{near}. If the path from q_{near} to q_{new} is collision free, q_{new} is added to the tree.

4.2 Rapidly Exploring Random Trees

It is possible to use the PRM planner as a single-query path planner by performing both the roadmap construction and the graph search in the query phase. However, if the start and goal configurations for the robot are known, this information can be used to explore $\mathcal{C}_f$ more efficiently. The rapidly-exploring random tree algorithm (RRT) is a simple, yet efficient algorithm, for building search trees that quickly explores $\mathcal{C}_f$. This makes it a suitable basis for an efficient single-query path planner [34, 38, 39].

4.2.1 Constructing RRTs

The BuildRRT-algorithm (algorithm 4.2.1) repeatedly expands a tree, T, from an initial configuration, q_{init}. The expansion step (illustrated in figure 4.4) is performed by picking a random point, q, in the environment that serves as a direction for the expansion, and locating the point already in the tree that is closest to q. This point, q_{near}, is used as the base for the expansion and a small step of length d is taken in the direction of q to a new point q_{new}. If the local planner is able to find a path from q_{near} to q_{new} and that path is collision-free, q_{new} is added to the tree as a child to q_{near}.

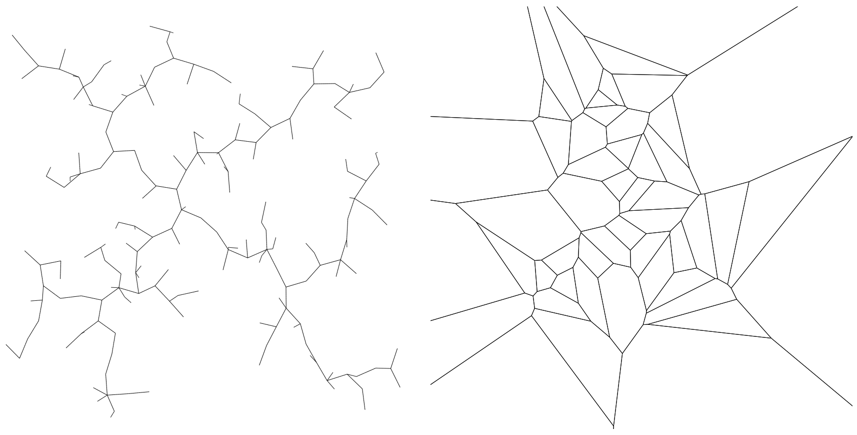
The RRT expansion gives a good sparse coverage of an area as can be

Algorithm 4.2.1 Builds an RRT from q_{init}.

```
NewState($q_{\text{near}}, q$)
  1  if $|q - q_{\text{near}}| \leq d$ then
  2     return $q$
  3  else
  4     return $\frac{d}{|q_{\text{new}} - q_{\text{near}}|}(q_{\text{new}} - q_{\text{near}})$
  5  end if

Extend($T, q$)
  1  $q_{\text{near}} \leftarrow$ closest-node($T, q$)
  2  $q_{\text{new}} \leftarrow$ NewState($q_{\text{near}}, q$)
  3  if $\gamma_c(P_L(q_{\text{near}}, q_{\text{new}}))$ then
  4     $T$. add-child($q_{\text{new}}, q_{\text{near}}$)
  5  end if

BuildRRT($q_{\text{init}}$)
  1  $T \leftarrow \{q_{\text{init}}\}$
  2  for $i = 0$ to $n$ do
  3     pick $q$ randomly from $\mathcal{C}$
  4     Extend($T, q$)
  5  end for
```



(a) RRT (b) Voronoi diagram

Figure 4.5: An example of an RRT in an empty environment with 200 nodes and the corresponding Voronoi-diagram after 50 expansions.

seen in figure 4.5(a). This is due to the method for choosing the node in the tree from which the next extension is made. In figure 4.5(b), a Voronoi diagram is shown for the first 50 nodes in the tree. Since the choice is done by finding the node closest to a random configuration, the probability of selecting a specific node is proportional to the size of the corresponding region in the Voronoi diagram. As can readily be seen in the diagram, this steers the algorithm to mainly expand from the outmost nodes, with large Voronoi regions, during the early stages of the tree construction.

4.2.2 Using RRTs for Path Planning

The RRT-Connect planner (algorithm 4.2.2; [34]) is a bidirectional planner that extends two RRTs from the initial configuration, q_0, and the goal configuration, q_g. The algorithm alternates between extending the two trees, and for each new node added to a tree, an attempt is made to also connect the node to the other tree. This connection attempt is made by extending from the node repeatedly until the nearest node in the other tree is reached or the extension fails.

A number of variations of the algorithm are also suggested in [34], e.g., the Connect-call can be replaced by a call to Extend which gives a somewhat simpler planner, or the planner can be made greedier by using Connect rather than Extend in all places.

4.2.3 Theoretical Results

Like the PRM algorithm, the RRT algorithm is probabilistically complete. This property is in fact more natural for RRTs since it is achieved solely by letting the RRT expansion continue until a solution is found. For the PRM algorithm, probabilistic completeness is only achieved by repeatedly expanding the roadmap when a runtime planning query fails, which negates the advantage of having a precompiled roadmap generated offline.

In [34], it is shown that if $\mathcal{C}_f$ is an open connected component of $\mathcal{C}$, the vertex distribution of an RRT converges to the random distribution used for building the RRT (which is usually uniform over the configuration space). The proof is constructed in two steps. First, it is shown that if

Algorithm 4.2.2 — Builds two trees from the initial configuration q_{init} and the goal configuration q_{goal} and continuously attempts to connect them.

```
CONNECT($T, q$)
 1  $q_{\text{near}} \leftarrow$ closest-node($T, q$)
 2  $q_{\text{new}} \leftarrow$ NEWSTATE($q_{\text{near}}, q$)
 3  while $\gamma_c(P_L(q_{\text{near}}, q_{\text{new}}))$ do
 4     $T$. add-child($q_{\text{near}}, q_{\text{new}}$)
 5     if $q_{\text{near}} = q_{\text{new}}$ then return
 6     $q_{\text{near}} \leftarrow q_{\text{new}}$
 7     $q_{\text{new}} \leftarrow$ NEWSTATE($q_{\text{near}}, q$)
 8  end while

RRTCONNECTPLANNER($q_{\text{init}}, q_{\text{goal}}$)
  1  $T_a \leftarrow \{q_{\text{init}}\}$
  2  $T_b \leftarrow \{q_{\text{goal}}\}$
  3  for $i = 0$ to $n$ do
  4     $q \leftarrow$ random free configuration
  5     if EXTEND($T_a, q$) succeeds then
  6        if CONNECT($T_b, q$) reaches $q$ then
  7           return make-path($T_a$. branch-of($q$), reverse($T_b$. branch-of($q$)))
  8        end if
  9     end if
 10     swap($T_a, T_b$)
 11  end for
```

an RRT is built in an open convex connected component, the RRT expansion will come arbitrarily close to any configuration in that component, given a sufficient number of iterations. This result is combined with the observation that for a connected component, there is a path connecting any two nodes, and this path can be covered with a finite number of open balls, which can be used to show that the RRT will cover also non-convex components. This result forms the basis of the probabilistic completeness proof, since any free configuration not lying on the boundary of an obstacle is in the center of a collision free ϵ-ball, and since the RRT is dense in $\mathcal{C}_f$, if the sampling distribution is, some vertex of the RRT will eventually fall in that ϵ-ball.

4.2.4 Optimization of paths

The selection of the tree node from which the extension will be made in BuildRRT is based on which node is closest to the randomly picked point in $\mathcal{C}_f$, according to some metric. This is a greedy heuristic that gives a rapid exploration of the configuration space as noted above. However, the paths that are found by the RRT planner are often not near the optimal solution.

The greediness of the algorithm can be reduced by also taking into account the path cost from the initial configuration to the nodes in the tree, similarly to how optimality is achieved for A*. This approach is briefly discussed in [9] where the following formula defines a heuristic cost for selecting what node to expand from.

$$c = \beta \, \text{length}(\text{path}(q_{\text{init}}, q_{\text{near}})) + \text{distance}(q_{\text{near}}, q) \tag{4.7}$$

The gain-factor, β, controls to what degree the path length influences the choice of node for expansion. For $\beta = 0$, the choice is made in the same manner as in the ordinary algorithm, but when β is increased a greater weight is placed on avoiding sub-optimal paths. However, when β reaches 1 (which in some sense is analogous to A*), all expansions will be from q_{init} since any other node would mean a detour. An appropriate choice of β is highly domain-dependent. In a case of repeated replanning, the authors of

[9] suggests a scheme where β initially is set to 0, and then continuously adjusted in the interval $[0, 0.65]$ depending on the success of the planner.

4.2.5 Probabilistic Roadmaps of Trees

In [1], a combination of the PRM and RRT algorithms, called probabilistic roadmaps of trees (PRT), was introduced. It combines the superior coverage of the RRT algorithm with some of the advantages of the PRM algorithm, e.g., a roadmap that can be reused for multiple queries and a greater degree of robustness. Like the PRM algorithm, the PRT algorithm generates a roadmap in a precompilation phase, but instead of trying to connect each node to its closest neighbors directly with a local path planner, a tree is constructed at each randomly sampled point. This tree can be an RRT or some other form of tree suitable for path planning.

The roadmap construction of the PRT algorithm proceeds in the following steps: First, a number of points in the configuration space are sampled randomly, and from each of them a tree is built. From these trees, a set of candidate edges are formed by taking the pairs of trees that lie closest together, and a number of randomly chosen pairs of trees. To add a candidate edge to the roadmap, nearby vertices in the two trees are tested for connection, or if that fails, a bidirectional path planning attempt is made. If this connection succeeds, an edge is added between the two trees. In [1], only edges that contribute to increasing the connectivity of the roadmap are added. A comparison in [1], showed better results for the PRT planner than both PRM and RRT planners on several hard problems.

4.3 Kinematic constraints

Up to this point, only holonomic robots have been considered. For many practical applications, the robots are subjected to various nonholonomic constraints. In this section some work related to kinematic constraints is presented, mainly studies of path planning for car-like robots. All these studies are done in a PRM setting, but similar techniques could also be applied with an RRT planner.

4.3.1 Kinematic Path Planning with PRM

The PRM algorithm can be used directly for path planning under kinematic constraints if it is equipped with a suitable local path planner. Svestka and Overmars introduced such a planner for general car-like robots and forward-moving cars, with a local path planner that produces paths by a combination of circular paths of minimum turn radius and straight lines [42].

Some care has to be taken for robots with an asymmetric local path planner, e.g., forward-moving cars, since the existence of a path in one direction does not necessarily imply that there exists a path in the other direction. Thus, it is necessary to use a directed graph for describing the roadmap.

To reduce the size of the roadmap, an acyclic graph was used with this planner. Such a graph can be generated in a simple and efficient manner for the general car (that can go both backwards and forward) since an undirected graph can be used in that case. For a forward-moving car, where a directed graph is required, it is nontrivial to retain an acyclic graph, but it can be achieved in terms of forward and backwards reachable sets [42].

4.3.2 Customizable PRM

The PRM algorithm spends a large amount of work to build a roadmap that can be used for answering multiple path planning queries for a certain robot in a given environment. This makes the planner rigid and slow in adapting to various changes to the planning problem, since information about both the environment and the robot mechanics is encoded into the roadmap.

A more flexible approach to PRM planning for car-like robots with regards to kinematic constraints was taken by Song and Amato [52]. Instead of building a roadmap for a specific car, an approximate roadmap is built, with only partial collision checking of edges. The idea is that this roadmap gives an approximate description of the environment that can be refined in the runtime phase for cars with a particular minimum turn radius.

The approximate roadmap of customizable PRM (C-PRM) is constructed in two steps:

1. A straight-line control roadmap is built, disregarding nonholonomic constraints, and with only midpoints of edges checked for collision.

2. The midpoints of the edges are used as nodes in the approximate roadmap, and an attempt is made to connect each node with the midpoints of the neighboring edges of the control roadmap. The connections are made with paths consisting of a straight-line segment and a circle arc of maximal radius, and the radius is recorded on the edge. No collision checking is done for these edges.

Planning for a car with a minimum turn radius, r, is done by first removing all edges with turn radius smaller than r. The planning is then performed similarly to the LazyPRM algorithm since full collision checking of the roadmap edges were not performed during roadmap construction.

4.3.3 Multi-Level Handling of Nonholonomic Constraints

For robotic systems subject to many nonholonomic constraints, it may be unfeasible to find a solution by direct application of the PRM algorithm. Such problems are treated in [49], where an algorithm called multi-level path planning is described. With this algorithm a simplified problem, P_0, with only holonomic and possibly some nonholonomic constraints, is solved first. The remaining nonholonomic constraints are added one at a time to form P_1, P_2 and so forth, until all nonholonomic constraints are handled. For each P_i, the solution to the previous problem, P_{i-1}, is used as a starting point that is refined so that the ith nonholonomic constraint is also satisfied.

For the refinement of a solution to satisfy additional constraints, two methods are proposed in [49]: the Pick and Link-method (PL) and the Tube Probabilistic Path Planner (Tube-PPP).

Using the PL method to transform a P_i solution into a P_{i+1} solution is done by first checking if the solution to P_i already satisfies all constraints of P_{i+1}. If that is the case, we are done. If some constraint is not satisfied, the PL-algorithm divides the local path into two shorter paths and tries to solve these with the extra constraints added. This operation is repeated until a solution that satisfies all constraints of P_{i+1} is found for each sub-path. The PL method will always find a solution if the local path planner fulfills what is referred to as the topological property in [48, 49]. The topological property states that for any ϵ, there is a η, such that for any q in a η-ball around a configuration q_0, there is a path from q_0 to q contained in a ϵ-ball around q_0. This means that the robot can move between any two nearby configurations without moving far from them, e.g., a car can move to a point close to the side by moving a short path forward and then backwards while turning appropriately. A thorough treatise of the topological property and its consequences for nonholonomic path planning is given in [48].

The other method for path refinement presented in [49] is Tube-PPP, which is a variation on the standard PRM algorithm, where the randomly generated configurations are picked from a tube around the solution to the P_i-problem. Tube-PPP generally finds paths faster and of higher quality than PL, but is only probabilistically complete, whereas PL is complete.

4.4 Dynamic Constraints

For real world applications, the magnitude of actuator forces are normally limited, and the planner needs to produce plans that stay within these bounds. To achieve this, plans are generally formulated so that they describe the development of the complete system state over time, in the form of a time-parameterized path through the state space. This stands in contrast to planning under holonomic and kinematic constraints, where the plan can be represented as a path through the configuration space of the robot.

Most of the work on dynamic motion planning with sampling-based algorithms has been done with RRTs. The necessary adaptations for handling dynamic planning in this framework is described in the first subsection. After that, applications to both simulated and physical robots are described.

4.4.1 RRTs for Kinodynamic Motion Planning

RRTs are well suited for motion planning under dynamic constraints, and were first introduced in a kinodynamic setting [38]. The RRT algorithm from section 4.2 uses a local path planner for connecting configurations just like the PRM algorithm, but connecting two arbitrary states in this manner is a difficult problem in itself for many dynamic systems. The advantage of tree-based planners, e.g., RRT planners, over roadmap-based planners is that they can be modified so that they do not require a local path planner that solves this problem. Instead, the tree can be built from the initial state by integrating the system dynamics under different control inputs.

This is done in the RRT algorithm by replacing the call to NewState in algorithm 4.2.1 and 4.2.2 with a NewState function that integrates the system dynamics under an applied control input. This function selects an appropriate control input in order to take the robot closer to the random state. If no such function is available, the planner can test different control inputs randomly and select the one that leads to the state closest to the new random state and add that to the tree.

The bidirectional RRT-Connect planner described in section 4.2.2 can be adapted to kinodynamic planning by using the above mentioned change [39, 40]. For the tree grown from the goal state, the expansion is done through integrating the system dynamics in reverse. However, there is a complication in that the two trees cannot be directly connected if there is no local planner available. In [39, 40], the two trees are grown until one node from each tree falls within a small distance, ϵ, from each other, and the remaining discontinuity is left for the controller to handle.

If such a discontinuity is not acceptable a number of solutions to this problem are proposed in [40], e.g., one or both of the trajectories can be slightly perturbed or classical shooting techniques can be applied. A detailed description of a method for connection through perturbations can be found in [18].

A proof of probabilistic completeness for the RRT-Connect planner for kinodynamic planning is given in [40].

4.4.2 Application to Autonomous Helicopters

In [16, 17, 19] a hybrid controller is described that is designed for aggressive maneuvering of an autonomous helicopter. The controller is based on a finite automaton, which is called the maneuver automaton. This automaton consists of states representing trim trajectories, which are trajectories with constant control inputs, and state transitions describing finite-time maneuvers for switching between different trim trajectories. The maneuver automaton provides a near-optimal controller for bringing the helicopter to an equilibrium point from any state of the system in obstacle-free environments.

For planning in the presence of obstacles, an algorithm similar to the RRT algorithm is used. First, an attempt is made to connect the initial state directly to the goal state using the maneuver automaton. If this is not successful, the segments of the path up to the first offending segment serve as an initial planning tree. Additional primary nodes are sampled from the set of equilibrium points, and the tree is extended to them with the hybrid automaton.

Since the primary nodes are equilibrium points, paths that go through these points are often far from optimal. To remedy this, additional secondary nodes are added to the tree. Each secondary node is placed at a random location along a trajectory segment and a connection attempt is made to the endpoint of the following segment, thereby bypassing an intermediary equilibrium point.

4.4.3 Other Applications

There are many examples of applications of RRTs for kinodynamic planning in the literature. In [39], a number of kinodynamic motion planning problems are presented, including translating and rotating bodies in both two and three dimensions, with bounded acceleration. In the small-scale environment (15-20 obstacles) in which the experiments were performed, the RRT-planner managed to solve the problem with a computation time ranging from seconds to minutes.

More realistic examples can be found in [12], where motion planning for a simulated car is presented. Unlike the path planning examples in section 4.3, a much more realistic model of the car dynamics is used here including slipping and skidding. This paper also presents an application of RRTs for planning the reentry flight path in a simulation with a prototype model of a reusable launch vehicle, NASA X33.

4.5 Time and Change

So far, we have discussed path and motion planning in static environments. Changes in the environment can be analyzed on two different time scales depending on how fast the changes occur. For rapidly changing environments, time has to be explicitly handled by the planner, something that may not be necessary for slower changing environments where the execution time of a plan is short compared to the speed of change. However, even for slowly changing environments, changes in the environment must be considered if the planner keeps data structures that represent the environment, e.g., roadmaps.

In this section, the explicit representation of time for moving obstacles will be considered first, before some methods for keeping roadmaps updated in changing environments.

4.5.1 Moving obstacles

If the trajectories of moving obstacles are known in advance, it is straightforward to extend sampling-based motion planning algorithms to also include moving obstacles (see, e.g., [35, 37]). This can be done by planning the path in the space-time: $C_f \times T$, where T is a time interval in which the plan will be contained. Stationary obstacles are represented as cylinders in $C_f \times T$ with a base of the shape that the obstacles have in C_f. For moving obstacles, the cylinders are slanted in the direction of motion. Sample based planners can be used in roughly the same way as in the timeless case, by sampling in $C \times T$ and testing connections against the space-time obstacles, but naturally paths are only possible forward in time. Since kinodynamic planners generally already incorporate time, they are often capable of handling moving obstacles of this type. This is the case with the application to autonomous helicopters described in section 4.4.2

4.5.2 Roadmap Updates

Planning in a changing environments with multiple-query algorithms, e.g., the PRM algorithm, involves a further complication since trajectories for the moving obstacles need to be known already when the roadmap is built. This is an unrealistic assumption for most real-world applications, where these obstacles may be detected first at query time. One solution is to use single-query algorithms, but for many path planning problems, the robot operates in a largely static environment with only a few non-stationary objects that the robot must avoid. In such cases, it can be of great benefit to have a precompiled roadmap for the static environment, so the online algorithm only needs to process moving obstacles. However, this requires a method for keeping the roadmap up to date in the changing environment.

In [26], a planner is described that generates a roadmap that only considers static obstacles. In the query phase, new obstacles may have appeared that invalidate some of the edges in the roadmap. Thus, it is necessary to check that the solution is still collision free. If it is not, the roadmap is partitioned into three components: nodes reachable from the start configuration, nodes reachable from the goal configuration, and nodes reachable from neither of the query nodes. An attempt is then made to find connections between these three components. This is done by first locating the nodes in the unreachable and goal component that are closest to the start configuration, and try to connect them to their closest neighbors in the start component. The nodes closest to the goal node in the start and unreachable component are likewise tried for connections with nodes in the goal component. If no direct connection is found, connection attempts are also made with an RRT-Connect planner.

Another method is described in [27]. Here the roadmap is complimented with a grid covering the same area of the workspace, and for each grid cell, a list is kept of all nodes and edges overlapping that cell. This makes it possible to invalidate regions of the roadmap quickly, as obstacles move through the workspace. When an obstacle enters a grid cell, all nodes and edges in that cell are temporarily invalidated until the object has left the cell. During a search in the query phase, only nodes and edges that are still valid are used.

If the possible locations of non-stationary obstacles are known during the roadmap construction it is possible to build a roadmap that guarantees a solution regardless of where the obstacles are located runtime if such a solution exist. An algorithm that does this is presented in [4]. This algorithm is similar to PRM algorithms for acyclic graphs in that it tracks the connectivity between the components of the graph. Since the connectivity varies as the obstacles are moved, connectivity is no longer a simple binary relation. Instead, the connectivity between the components of the roadmap is described with the set of obstacle positions for which two components are connected.

The algorithms in [4, 26, 27] are built on the assumption that the obstacles are stationary during plan execution. In another algorithm described in [5], obstacles moving along known trajectories can be handled. This algorithm also starts with a roadmap built for the stationary obstacles and the search through the roadmap is performed with an A*-like search algorithm. Since only the static obstacles are respected during the roadmap construction, collisions with moving obstacles are still possible when traversing an edge in the roadmap. When a moving obstacle passes through an edge, it gives rise to obstacle regions on the surface traced by the edge in the space-time. In order to find a fully collision free path, a grid search is performed on this surface, for each edge that is reached during the search. This search is performed intertwined with searches on other edges in the order specified by the A*-heuristic. A number of simulated experiments using this algorithm are described for both mobile robots in corridors and articulated robots in a 3D workspace. The algorithm can also be used for planning paths for multiple robots, by planning a path for one robot at a time.

Part II

Chapter 5

Path Planning Framework

In order to test how different path planning algorithms can be used for practical applications with an autonomous helicopter, a path planning framework has been developed. Within this framework, both PRM and RRT based planners have been implemented. The basic algorithms for these two planners were described in section 4.1 and 4.2, and in this chapter, a number of extensions of the algorithms are described in the context of the path planning framework. The most important extensions are related to more flexible ways of handling different types of path planning constraints, and it will be shown how constraints can be applied at various stages of the planning process.

This chapter begins with an overview of the path planning process. After that a discussion follows on how motion constraints and obstacle constraints can be applied at different stages during the planning process. The last section describes how the path resulting from the PRM and RRT algorithms can be improved by various smoothing procedures.

The path planning algorithms described in this chapter require a local path planner that closely matches the maneuvering capabilities of the real helicopter and a collision checker that can check that these paths are not in collision with the environment. The implementation of the collision checker and other obstacle constraints are described in the next chapter, while the different local path planners that have been used with the helicopter are described in chapter 7. That chapter also describes the integration of the

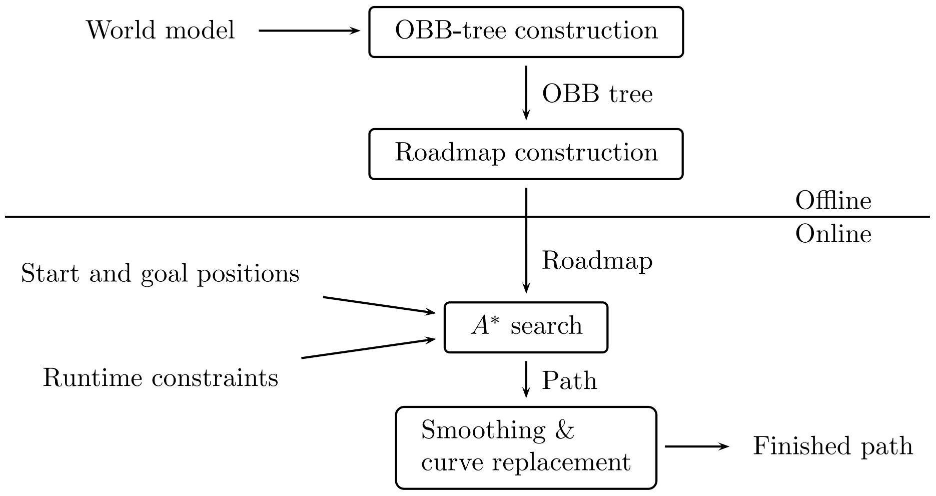


Figure 5.1: The main stages of path planning-process for the PRM planner.

path planner module with the helicopter system.

5.1 Path Planner Overview

The overall planning process is similar for the two planners, but for clarity of presentation, they are described separately in this section. The extensions described later in this chapter were first developed for the PRM planner which therefore will be described first, followed by an account of how they affect the RRT planner.

The main stages of the path planning process for the PRM planner are shown in figure 5.1. Before the roadmap is constructed, the environment is preprocessed with the OBBTree algorithm to enable efficient collision checking (described in the next chapter). The roadmap is constructed using the MakeRoadmap algorithm (algorithm 4.1.1), with the collision constraint implemented by the OBB-tree.

For the motion constraints of the helicopter, two different approaches are implemented as described in section 5.2. In the first approach, the roadmap is generated with local paths directly suitable for the helicopter,

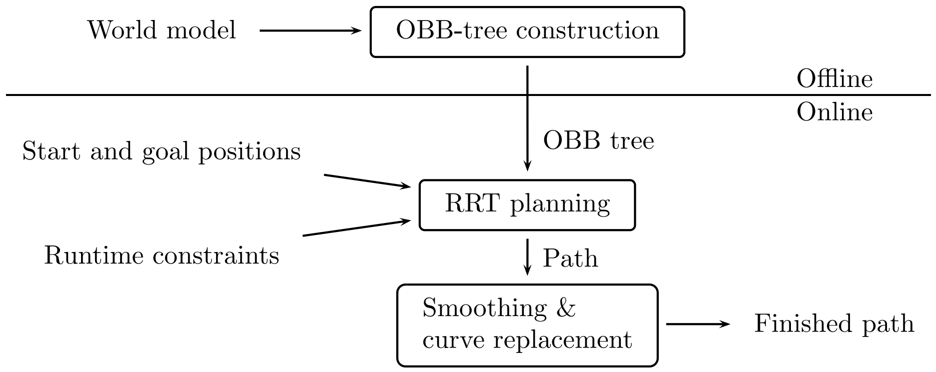


Figure 5.2: The main stages of path planning-process for the RRT planner.

while in the second approach, the motion constraints are postponed to a later stage and straight-line paths are used for connecting the nodes.

In the runtime phase, the start and goal configurations are connected to the graph and a solution path is found through graph search as with the standard PRMQuery algorithm (algorithm 4.1.2). Unlike the standard PRM algorithm, this implementation makes it possible to add further constraints in the runtime phase. This extension is described in section 5.3.

As a last step, the path is subjected to a series of post processing steps for improving the quality of the plan. If the motion constraints were disregarded during the roadmap construction, the plan is also transformed into a piecewise cubic curve at this stage.

The RRT planner is an implementation of the RRT-Connect planner (algorithm 4.2.2) and constructs two RRTs from the start and goal configurations. The path planning process is similar to that of the PRM planner and is shown in figure 5.2. The main difference is that the trees are constructed in the runtime phase, instead of in the offline phase as is the case with the roadmap graph of the PRM algorithm. This means that for the RRT planner, there is no need to distinguish obstacle constraints that can be evaluated offline from those that need to be evaluated during runtime. For motion constraints, both of the approaches that were described above can be used.

Even if the RRT planner operates fully in the runtime phase, the OBB-

Tree algorithm requires a significant amount of time to preprocess larger geometric models. For a truly flexible planner, the OBBTree algorithm has to be replaced with an incremental collision checker that is able to respond quickly to changes in the environment.

5.2 Motion Constraints

The basic PRM and RRT algorithms only consider obstacle constraints, while kinematic constraints can be handled with a suitable local path planner. In this section, we will look at two different ways of handling dynamic constraints with the PRM and RRT algorithm.

5.2.1 State Space Roadmap

The most straightforward way of handling dynamic constraints in the PRM and RRT algorithms is to complement the configurations with their derivatives and record the complete state at each node in the graph. This enables the local path planner to adapt the path between two nodes to their associated derivatives, which is necessary to respect the dynamic constraints at boundary points between adjacent edges in the solution path.

The drawback of this approach is that the dimensionality of the space in which the roadmap is situated is doubled. In the implemented planner, only the direction of flight is recorded at each node and a fixed magnitude is used for the derivative vectors. Even so, the three dimensional position and the direction of flight vector result in an increase from three to five dimensions.

In addition to an increased number of dimensions, this method also requires a directed graph for a roadmap, since the path between two nodes is different depending on if the helicopter flies from the first node to the second or in the other direction. Extending the basic PRM algorithm to handle directed graphs is straightforward, and only involves testing the local path in both directions and adding edges for the directions in which the paths satisfy the no-collision constraints. This is done by duplicating line 3–6 in ConnectNode (algorithm 4.1.1), with q and n reversed.

5.2.2 Multi-level Path Planning

An alternative approach to nonholonomic planning is to postpone the nonholonomic constraints to the runtime phase. Sekhavat, Svestka, Laumond and Overmars developed a path planner for a car-like robot pulling a number of trailers that follows this approach, which was described in section 4.3.3 [49]. With their multi-level planner, the problem is first solved with only the nonholonomic constraint that arises from the car itself. The solution to this relaxed problem is then refined by adding the non-holonomic constraints for the trailers one at a time. For locally controllable robots, it is always possible to transform the solution so that the non-holonomic constraints are also respected as long as there exists a non-zero margin between the obstacles and the solution to the relaxed problem.

Path planning for a helicopter is somewhat different than for locally controllable robots, but a similar approach can still be used. Since helicopters are free-flying robots that can follow any path, the output from a holonomic path planner can be used directly, e.g., the PRM algorithm can be used with a straight-line local path planner to produce piecewise linear paths. However, if the dynamics are completely ignored, the paths produced by the path planner will be of low quality, since the curvature determines the speed at which the helicopter can traverse it. A piecewise linear path requires the helicopter to stop and hover at each waypoint along the path in order to change direction of flight. Even if the piecewise linear path is unsuitable, it can serve as a good starting point for finding paths for the helicopter. In order to better respect the dynamic constraints of the helicopter during flight at higher speeds, the sharp corners must be eliminated. With the multi-level approach for helicopters, described below, this is done by replacing the straight-line path segments with Bézier curves.

The replacement procedure is shown in algorithm 5.2.1, and an example of how the path is transformed for the helicopter is shown in figure 5.3. Figure 5.3(a) shows a piecewise linear path that could be the result of a graph search. In the first step, the configurations along the path are replaced with state descriptions. This replacement is made by associating a

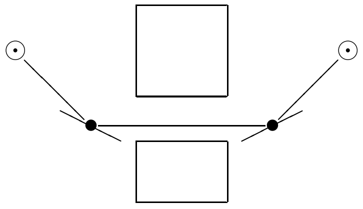
Algorithm 5.2.1 Path Augmentation Algorithm

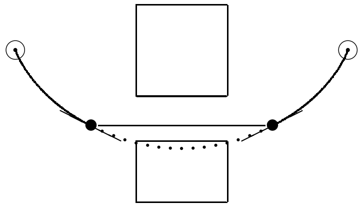
```
CalculateDirections($\langle p_0, \ldots, p_n \rangle$)
1  $dp_0 = \frac{p_1 - p_0}{|p_1 - p_0|}$
2  for $i = 1$ to $n-1$ do
3      $dp_i = \frac{p_{i+1} - p_{i-1}}{|p_{i+1} - p_{i-1}|}$
4  end for
5  $dp_n = \frac{p_n - p_{n-1}}{|p_n - p_{n-1}|}$

AlignDirectionsToLinear($\langle p_0, \ldots, p_n \rangle, \langle \pi_0, \ldots, \pi_n \rangle$)
1  for $i = 1$ to $n-1$ do
2      if linear($\pi_{i-1}$) $\wedge$ cubic($\pi_i$) then
3          $dp_i = \frac{p_i - p_{i-1}}{|p_i - p_{i-1}|}$
4      else if cubic($\pi_{i-1}$)$\wedge$linear($\pi_i$) then
5          $dp_i = \frac{p_{i+1} - p_i}{|p_{i+1} - p_i|}$
6      end if
7  end for

ReplaceCurves($\langle p_0, \ldots, p_n \rangle, \langle dp_0, \ldots, dp_n \rangle, \gamma_c$)
1  for $i = 0$ to $n-1$ do
2      $\pi = P_{\text{cubic}}(\langle p_i, dp_i \rangle, \langle p_{i+1}, dp_{i+1} \rangle)$
3      if $\gamma_c(\pi)$ then
4          $\pi_i = \pi$
5      end if
6  end for

AugmentPath($\langle p_0, \ldots p_n \rangle, \gamma_c$)
1  CalculateDirections($\langle p_0, \ldots p_n \rangle$)
2  ReplaceCurves($\langle p_0, \ldots, p_n \rangle, \langle dp_0, \ldots, dp_n \rangle, \gamma_c$)
3  AlignDirectionsToLinear($\langle p_0, \ldots p_n \rangle$)
4  ReplaceCurves($\langle p_0, \ldots, p_n \rangle, \langle dp_0, \ldots, dp_n \rangle, \gamma_c$)
```


(a) Linear path.


(b) Primary attempt of path transformation.

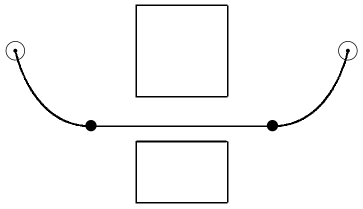

(c) Secondary attempt of path transformation.

Figure 5.3: Transformation from linear path segments to cubic path segments required for smooth flight.

direction of flight vector to each node that is parallel with a line connecting the two neighbor nodes (shown as a short line on the intermediary nodes). For each segment, an attempt is made to replace the straight lines with a cubic curve matching these vectors (figure 5.3(b)). For segments where this is not possible, e.g., the middle segment in figure 5.3(b), the linear path will remain, but an additional attempt to achieve smooth transitions is made by trying to align the neighboring curves to the straight-line path (figure 5.3(c)). If this step also fails, a sharp corner is left in the final path, and the helicopter must stop and hover at this point in order to realign for the next segment. In practice, this happens infrequently as we will see in section 9.4, and normally only in cramped environments.

5.2.3 Related Work

Planning in the robot's state space is the standard method for motion planning under dynamic constraints and is used in all of the planners described in section 4.4 [12, 16–19, 38–40]. Here, a slightly different approach has been taken. No exact description of how the robot state is evolving over time is derived. Instead, a geometric path is planned in $\mathcal{W}$, taking dynamic constraints into account to make it efficiently traversable by the helicopter. This detachment of the path planner from the low-level control system has both advantages and disadvantages compared to planning the robot's motions directly.

From a system design point of view, it provides a better separation of concern. The path planner is designed with a simpler model of the robot, and a higher-level, intermediary layer of the control system executes the appropriate control signals for following the path from the planner; this control system architecture is discussed in more detail in chapter 7. The main drawback is that the full capabilities of the robot may not be utilized. To achieve that, a tighter coupling between the planner and the control system of the robot is needed. Such an integrated motion planner and control system approach has been developed by Frazzoli with a maneuver automaton that closely models the capabilities of an autonomous helicopter [16, 17, 19].

Another difference compared to the above-mentioned planners is that they all are based on RRTs. However, unlike most of those applications, an important goal with this work has been to implement a planner that works well in large-scale realistic environments. As we will see in section 9.4 such environments may require some preprocessing to enable the planner to find paths quickly and reliably, which is why the PRM algorithm is also interesting for this type of applications.

To reduce the size of the roadmap for these large environments, the multi-level technique was borrowed from the work on kinematic path planners for car-like robots [49]. The requirements on a planner for a point robot with dynamic constraints are similar to planners for car-like robots, since the path curvature is constrained in both cases. Thus the multi-level technique can be useful also for planning paths for a helicopter, and makes it possible to achieve smooth flight paths also from a roadmap with straight-line edges.

5.3 Runtime Constraints

During the construction of the roadmap, each edge is checked for collision. In many applications there are other types of constraints on the plan, e.g., the operator may want to specify a maximal flight altitude for the helicopter or areas that the helicopter is forbidden to enter. Such geometric constraints can be handled in the same manner as the no-collision constraint, γ_c, by replacing it with the conjunction of γ_c and the additional constraints.

With the standard PRM algorithm, all constraints added in this way must be known in advance, before the roadmap is constructed, leading to an inflexible planner. However for practical applications, conditions are often changing and the path planner must be able to adapt rapidly to these changes. In this section, an extension to the PRM algorithm is presented that allows additional constraints to be specified in the runtime phase. With this extension, it is possible for the operator to influence the path planner with new constraints in the runtime phase while still profiting from the work invested in the roadmap construction.

Algorithm 5.3.1 The PRMQuery-algorithm with runtime-constraints.

```
ExpandUnderConstraints($n, G, \Gamma_r$)
1  return $\{n' \mid n' \in$ Expand$(n, G), \forall \gamma \in \Gamma_r : \gamma(n')\}$

PRMQuery($q_0, q_g, \Gamma_o, \Gamma_r, P_L, G$)
1  ConnectNode($q_0, \bigwedge(\Gamma_o \cup \Gamma_r), P_L, G$)
2  ConnectNode($q_g, \bigwedge(\Gamma_o \cup \Gamma_r), P_L, G$)
3  ExpandNodeFn $\leftarrow \lambda(n, G)$ : ExpandUnderConstraints($n, G, \Gamma_r$)
4  return  A*-search ($q_0, q_g, G$, ExpandNodeFn)
```

5.3.1 PRM Planner with Runtime Constraints

The set of planning constraints are divided into offline constraints, Γ_o, that are known in advance, and runtime constraints, Γ_r, that are not known until the query phase. A roadmap is generated for the offline constraints in the same way as with the standard PRM algorithm (with $\bigwedge \Gamma_o$ replacing γ_c in algorithm 4.1.1).

The PRMQuery algorithm (algorithm 4.1.2) has to be altered in two places for handling new constraints in the query phase; the altered version is shown in algorithm 5.3.1. The start and goal configurations, q_0 and q_g, are added as before, but the connections are checked against both the offline constraints and the runtime constraints.

The second modification to the algorithm concerns the graph search. The edges in the graph are already known to satisfy the constraints in Γ_o, but may violate constraints in Γ_r. To guarantee that the returned path satisfies all constraints, each edge is checked against the runtime constraints during node expansion, and only nodes reached through edges satisfying all the constraints in Γ_r are added to the search queue.

5.3.2 Implemented Runtime Constraints

The runtime constraints must be computed quickly and therefore simple constraints are most suitable for use in this manner. Here follows some examples of constraints that have been implemented and tested with the path planner; similar types of constraints can easily be plugged into the planner:

Min and max altitude limit the altitude of the helicopter.

No-fly zones are horizontal polygons covering an area, over which the helicopter is not allowed to fly.

Generic obstacles include any 3D-object consisting of polygons.

Even if the first two types of constraints can also be represented by the more general third type, it is still useful to treat them separately since they represent common types of constraints that the user may want to place on the planner. This makes it easier for the user to specify them and can also improve the performance since more efficient special-purpose algorithms can be used. The implementations of the different constraints are described in chapter 6.

5.3.3 Repairing Broken Roadmap Connectivity

The introduction of further constraints in the query phase may break the connectivity of the roadmap. To alleviate this problem, an optional RRT connection step was added to the path planner similarly to [26]. If the graph search fails, the graph is partitioned into two smaller graphs, containing the reached and unreached nodes respectively, and one node is selected from each partition. From the set of reached nodes, the node, n_1, closest to the goal node is picked, and from the set of unreached nodes, the node, n_2, closest to n_1 is picked. An attempt is then made to connect these two nodes with the RRT-Connect planner. If this connection attempt is successful, the intermediary nodes in the solution are added to the roadmap and a new graph-search is attempted. This process continues until a path is found between the start and goal configuration, or the RRT connection attempt fails.

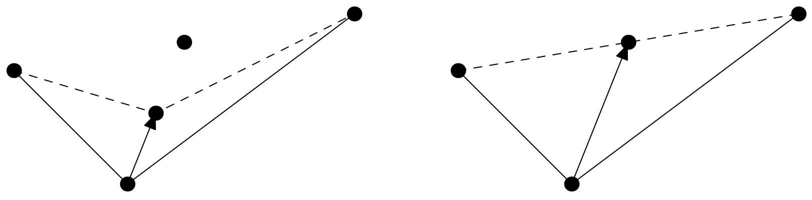
A side effect of this algorithm is an improved ability to connect the start and goal configuration to the roadmap. If a direct connection cannot be found between the start configuration and the roadmap, the graph search will fail instantly and an RRT connection is attempted, which significantly improves the chance of finding a connection. The same situation occurs if the goal configuration cannot be directly connected. In this case, the search fails to find the goal, and an attempt is made to connect the closest node in the roadmap to the goal configuration with the RRT planner.

5.3.4 Related Work

The basic PRM algorithm requires all constraints to be known at roadmap compilation time. During recent years some effort has been invested in combining precompiled roadmaps for a static environment with online planning for moving obstacles. A number of such planners were described in section 4.5 and the work on runtime constraints described above is another example.

The RRT-reconnection step was first implemented in a planner by Jaillet and Siméon [26]. That planner uses an acyclic roadmap, which makes a reconnection strategy vital, since any lost edge due to new obstacles breaks the connectivity of the roadmap. This stands in contrast to the PRM planner that has been used with the WITAS helicopter, where dense cyclic roadmaps have been used. This makes a reconnection step much less important since new runtime constraints rarely affect enough edges to break the roadmap connectivity. A study of the usefulness of the RRT-reconnection step in this setting is given in section 9.3.2. However, the main motivation for using dense roadmaps is that they lead to better plans, which is something that also is shown in chapter 9.

Another difference between the planners is that Jaillet and Siméon's planner uses lazy collision checking like the LazyPRM algorithm, while the runtime constraints described above are checked during the graph search. Lazy collision checking is advantageous when collision checking is expensive since a minimal number of checks are performed. If the constraints are cheap to check, e.g., simple no-fly zones or altitude limits, the graph search may be the most expensive part of the online planning process, and in such cases checking the constraints during the search can improve performance.



(a) Alignment halfway. (b) Full alignment.

Figure 5.4: Alignment of nodes.

5.4 Post Processing the Plan

The random nature and limited density of the sampled nodes make the PRM and RRT algorithms produce paths that are often jagged and irregular with occasional detours. In order to improve the quality of the paths, a smoothing step is usually added to these planners [29, 41].

For the implemented path planner the following smoothing steps are performed:

Alignment For each node along the path, two attempts are made to move it to a point that straightens out the path. The new locations that are tested for the nodes are illustrated in figure 5.4. For both attempts, the point, m, in the middle between the two neighbors is located. In the first pass (figure 5.4(a)), an attempt is made to move the node, n, halfway to m, and in the second pass (figure 5.4(b)), an attempt is made to move it all the way to m.

Node Elimination For each node along the path, an attempt is made to eliminate the node by connecting the two adjacent nodes directly. If the connection satisfies all constraints, the middle node is eliminated.

For the multi-level planner, the curve replacement step described in section 5.2.2, is performed between the alignment and the elimination step.

Chapter 6

Obstacle Constraints

With the PRM and RRT algorithms, a local path planner is used for finding paths between nearby nodes in the graph or tree. This planner takes care of the intrinsic motion constraints of the robot and generates paths that the robot can follow, but before they can be added as edges to the graph or tree, they must be checked so that they also satisfy external constraints, e.g., that the paths are collision free. In this chapter, we will look at how these tests are performed for a number of constraints that are relevant for the helicopter path planning problem.

The most important of the external constraints to consider is the collision avoidance constraint. This constraint is extremely important for a helicopter path planner since any contact between the helicopter and an obstacle would likely lead to a crash. The environment in which the helicopter is operating can also be very complex with many irregular obstacles, and it is therefore necessary to have an efficient algorithm for testing if a path is collision free.

We will begin this chapter with a description of the environment that needs to be handled by the collision checker, before we move on to a description of the OBBTree algorithm, which has served as the collision checker in the path planner implementation. After describing the collision-checking algorithm, the chapter ends with an account of the different runtime constraints that have been used with the planner.

6.1 The Environment and Helicopter Model

In order to avoid collisions with obstacles, the path planner needs an accurate description of the environment. The main flight-test area that has been used in the WITAS project is a training area for rescue workers in Revinge in southern Sweden. The training area is used for practicing different types of rescue operations and contains a road network, several building structures, and a number of accident sites. This makes it very well suited for more advanced flight tests where operations over traffic and in urban areas are simulated.

Simulation of the helicopter and the environment has played an important role in the WITAS project, and for these purposes, a detailed 3D model of the Revinge flight test area has been constructed. This model has also been used as input to the path planner. The model consists of 205 different objects, mostly buildings and trees, but also masts, lampposts, fences etc., and elevation data for the ground with 1 m resolution. Together, these objects consists of almost 140 000 polygons.

The collision checker described in this chapter does not require knowledge of the individual objects, but instead treats the 3D model as an unordered set of polygons. The only exception is the elevation data that is handled separately. Currently, the path planner operates under the simplifying assumption that the model of the environment is known in advance.

The fuselage of the helicopter is approximately 2 m long, and the rotor is approximately 3 m in diameter. The reference point of the helicopter is centered near the rotor axis, which means no point of the helicopter is more than 2 m away from that point. This distance will be called the radius of the helicopter.

For safety resons, an extra margin may be enforced between the helicopter and any obstacle in the environment. The size of this margin must be adjusted for the sensor capabilities and control precision of the robot, and is currently set to 6 m for the RMAX helicopter. Since the helicopter is small in comparison with the obstacles in the environment and the safety margin, it can be modeled as a point object instead of a more complex 3D model. We will see later in this chapter that this approximation greatly simplifies the collision-checking algorithm. To maintain the safety margin when the helicopter is represented as a point, the margin is extended with the helicopter radius.

6.2 The Collision Checker

The collision-checking algorithm used in the path planning framework is based on the OBBTree algorithm [21]. The OBBTree algorithm constructs a tree of oriented bounding boxes (OBBs) around the obstacles in the environment by including all polygons in the root box and then recursively dividing the polygons into smaller and smaller boxes. This makes it possible to efficiently determine if a collision has occurred. In this section we will first look at how the individual OBBs are constructed and tested for collision before we turn to trees of OBBs.

6.2.1 OBBs

Choosing the alignment for OBBs is not trivial. One method is to apply principal component analysis[1] (PCA) to the vertices of the polygons to determine the axes of largest and smallest variation [21]. The OBB is aligned along these two axes together with a third axis chosen perpendicular to them and follows the main directions for most types of objects.

The OBBs are represented with the six planes that bound the box. These are found by iterating over the vertices of the polygons and projecting them onto the three axes. The bounding planes for one of these axes are constructed so that they are perpendicular to the axis and are located at the most extreme of the projected points along the corresponding axis.

To check if the helicopter intersects with an OBB, the OBB is enlarged with the safety margin plus the maximal radius of the helicopter. If the position of the helicopter falls on the inside of all six bounding planes, the position is considered a collision point for the helicopter.

In addition to test if individual points are collision free, the PRM and RRT algorithms also require the collision checker to test if a collision will

[1]PCA is done by finding the eigenvectors of the covariance matrix for a set of vectors. The eigenvectors corresponding to the largest and smallest eigenvalues give the direction of largest and smallest variation of the vectors.

occur if the helicopter flies along a path, τ. Since the helicopter is represented by a single point, it is enough to check if the path itself intersects any of the enlarged OBBs. This is done by locating the points where $\tau(s)$ intersects the bounding planes of the OBB. If any of these intersection points lie within the corresponding side of the OBB, the curve must intersect the OBB.

The curve describing the motion of the helicopter is given as a third degree polynomial with vector coefficients:

$$\tau(s) = \mathbf{a}_0 + \mathbf{a}_1 s + \mathbf{a}_2 s^2 + \mathbf{a}_3 s^3 \tag{6.1}$$

To find where the curve intersects the plane coinciding with one of the faces of the OBB, the polynomial is first projected on the normal of that plane, $\hat{\mathbf{n}}$, which gives a scalar polynomial of the form:

$$a_0 + a_1 s + a_2 s^2 + a_3 s^3, \quad a_i = \mathbf{a}_i \cdot \hat{\mathbf{n}} \tag{6.2}$$

Letting d be the orthogonal distance from the plane to the origin (with sign corresponding to the direction of $\hat{\mathbf{n}}$), the values of s for which $\tau(s)$ intersects the plane is given by

$$a_0 + a_1 s + a_2 s^2 + a_3 s^3 = d \tag{6.3}$$

This equation is solved analytically, and the solutions where $s \in [0, 1]$ are inserted into equation 6.1, which results in 0–3 points on this plane. These points are compared to the planes corresponding to the four neighboring faces of the OBB. If any point lies within the space confined by these planes, the curve intersects this face of the OBB and hence the OBB itself. If they all lie outside these planes, the curve does not intersect this side, and if all the sides are collision free and the path starts outside the OBB, the OBB is collision free.

6.2.2 Building the OBB tree

The OBBTree algorithm builds the tree by first constructing an OBB that contains all polygons in the model of the environment. This OBB is associated with the root node of the OBB tree.

The polygons in the OBB are then divided into two groups in order to create two sub nodes. This division is done by finding the middle point of the longest axis of the root OBB and forming a division plane at that point perpendicular to the axis. The polygons are divided into the two groups depending on which side their centers fall. A new OBB is then generated for each of the two subsets of polygons, and these OBBs are added to the OBB tree as children to the root OBB. The process of dividing the OBBs into smaller and smaller sub-boxes is carried on until the level of individual polygons is reached.

A separate OBB tree is generated for the ground data while the rest of the obstacles are represented by another tree. The tree constructed for the ground data is limited to a depth of 10, in order to reduce the size of the OBB tree. This gives a sufficient resolution for environments where the ground is flat, e.g., the flight-test area, but may be adjusted for other environments with different requirements.

6.2.3 Intersection with OBB Trees

Intersection tests with an OBB tree are done in the same way for both points and paths. The only difference is in the intersection test with individual OBBs.

When a point or path has to be checked for a collision, the algorithm looks for collisions recursively down the OBB tree. It starts by checking the OBB corresponding to the top node of the tree for a collision. If this check reports that there is no collision, there can be no intersection of the model by the point or path since the whole model is contained in this OBB. If there is a collision, the two subtrees have to be checked. This process continues recursively down to the leaves of the OBB tree. If a collision is reported for any of the leaves, the point or path is considered to be in collision with the OBB tree.

6.3 Runtime Constraints

A number of runtime constraints were briefly introduced in section 5.3.2. For each constraint an implementation is required for testing constraint satisfaction for both configurations, q, and paths, τ. These tests are implemented in the following way:

Min and max altitude: Configurations satisfy the constraint if q_z is above or below the min or max altitude respectively. Paths are tested by seeing if any s, for which τ intersects the plane at the specified altitude, is in the interval $[0, 1]$, and that the start point does not violate the constraint.

No-fly zones: The satisfaction test for configurations involves checking if the helicopter position, projected in the xy-plane, is within the polygon. This is done by extending a horizontal ray from the point in the x direction and counting the number of edges that the ray intersects. If this number is odd, the point is within the no-fly zone.

Paths are tested similarly to the path test for OBBs. For each plane bounding the no-fly zone, the intersections of τ is solved analytically. If any solution, s, where $s \in [0, 1]$, falls between the neighboring bounding planes, the constraint is violated.

Generic obstacles: Collision checking for generic obstacles is done by using OBB trees in the same manner as for collision checking with the environment. This means that an OBB tree must be generated for the obstacle, which makes this method usable only for moderately complex objects, if a fast response is required.

Chapter 7

System Integration

An important goal of this research has been to investigate how a sampling-based path planner can be used with a real physical helicopter. For this reason, the path planning algorithms developed within the framework of chapter 5 have been packaged in a path planning module and been integrated into the onboard system of the helicopter used in the WITAS project. These planners share the same public interface and interact with the rest of the system in the same manner, so the following description is equally valid for all the variations of the PRM and RRT algorithms that were described in chapter 5.

This chapter starts with an overview of the WITAS system architecture and the different software components that are used in conjunction with the path planning module. A more detailed description of the architecture can be found in [15]. One of the most important part of the system that directly affects the path planner is the controller used for moving between different points. Two control modes that have been used together with the planner are therefore described in the second section of this chapter. The last section gives examples of how the different parts of the system operate together to perform two tasks where the path planner is involved as an important component.

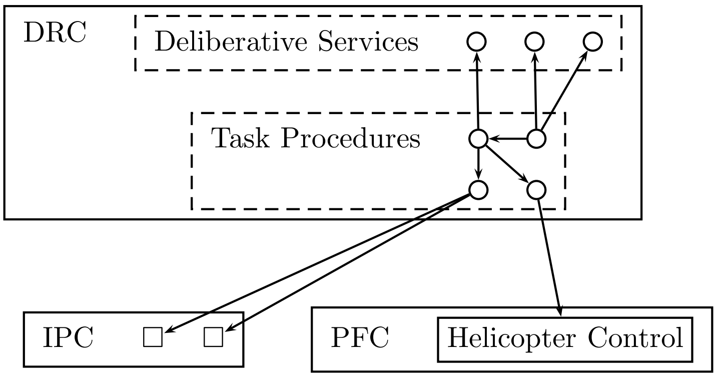


Figure 7.1: System architecture.

7.1 System Architecture

The physical platform is a Yamaha RMAX helicopter with a total length of 3.6 m and a maximum take-off weight of 95 kg. It is designed for easy operation by radio control, and is equipped with an attitude sensor system, YAS, and an attitude control system, YACS, which have also been used in developing autonomous flight modes. To provide all the required data for autonomous operation, the hardware has been complemented with an array of sensors, including a GPS, a pressure sensor, a compass and a video camera. The helicopter also carries a box on the side containing three PC104 computers on which all software required for autonomous operation runs. The three computers are each equipped with Intel Pentium III processors running at 700 MHz and 256 MB of memory, and are connected to each other with an Ethernet network. The software resides on 512 MB compact flash media on the computers.

The different software components of the system are distributed on the three computers according to the diagram in figure 7.1. The central components in the system that initiate all actions of the helicopter are the task procedures, TPs, that run on the Deliberative Reactive Computer (DRC). TPs are responsible for executing particular behaviors of the helicopter, such as monitoring an area, following a vehicle or just flying to a point. The behaviors described by the TPs can be of various degrees of complexity where higher-level TPs call TPs with simpler behaviors. TPs are reactive software components based on finite state machines, but with the added capability of running arbitrary functions on state transitions or in response to events.

For intelligent behavior, TPs make use of deliberative services, which are programs with procedures for reasoning or that encode different forms of knowledge. The path planner module is an example of such a deliberative service, and is responsible for finding a path connecting two given points, while the actual flight along the path is managed by a TP.

The helicopter controller runs on the primary flight computer, PFC. This computer uses the RTAI[1] real-time operating system to satisfy the hard real-time requirements of the control routines. The PFC is essential for maintaining stable flight of the helicopter and contains all the required software to keep the helicopter airborne. This can be contrasted with the DRC, where non-vital processes that add intelligence to the system are running in soft real time. The PFC also handles the direct communication with the helicopter platform and the various sensors.

The third computer is the Image Processing Computer, which is concerned with the vision capabilities of the helicopter. It is also responsible for controlling the camera platform.

The different components on the DRC are independent processes that communicate through CORBA[2]. This makes it easy to run the system in many configurations, and new components can easily be added to the system in a natural way. This software infrastructure also allows for a wide variety of programming languages to be used within the system, and makes it possible to choose the programming language that is most well suited for a particular task.

[1]Real-Time Application Interface, `www.rtai.org`
[2]Common Object Request Broker Architecture, `www.omg.org`

7.2 Local Path Planning and Control

The control system used on the WITAS platform is a hybrid control system [13]. It has a number of different control laws used for the different flight modes of the helicopter and uses hierarchical concurrent state machines, HCSMs, for mode switching between them.

The flight modes that are of primary concern for path planning are the modes for flying along specified curves. During the WITAS project two different control modes have been developed and used in conjunction with the path planner: the proportional navigation controller, PN, which was designed for dynamically locking on a stationary or moving point and the trajectory following controller, TF, that was designed for following a pre-specified curve and is the control mode that is currently used. Before these control modes are described, we will describe the curves that are used with both these controllers.

7.2.1 Curve Description

The PRM and RRT planners are adapted to specific robots by using a robot-specific local path planner. The local path planner connects two helicopter states with a path segment that has to match the curve form corresponding to some control mode of the robot. On the helicopter the trajectory-following control mode, TF, is used for following paths in the form of a cubic polynomial in three dimensions.

The choice of cubic curves for path representation was originally motivated by the proportional navigation controller, PN, which is described later in this chapter. The PN controller traces a curve that closely approximates a cubic polynomial when tracking a stationary target. Due to a number of good characteristics of this curve type, e.g., flexibility, possibility of obtaining C^2-continuity at end-points and analytical solvability (which is used for collision checking), it remained when the switch to the TF controller was made. The curve is described by the equation

$$\tau(s) = \mathbf{a}_0 + \mathbf{a}_1 s + \mathbf{a}_2 s^2 + \mathbf{a}_3 s^3 \tag{7.1}$$

parameterized on the interval $s = [0, 1]$. Given two helicopter states, $x_0 = (q_0, \dot{q}_0)$ and $x_g = (q_g, \dot{q}_g)$, the local path planner must construct a path of this form. Since τ has to match x_0 and x_g at its end points, we have

$$\tau(0) = q_0 \qquad \dot{\tau}(0) = c\dot{q}_0 \tag{7.2}$$
$$\tau(1) = q_g \qquad \dot{\tau}(1) = c\dot{q}_g \tag{7.3}$$

for some constant c. The constant, c, is currently chosen so that the magnitudes of $\dot{\tau}(0)$ and $\dot{\tau}(1)$ are equal to the distance between the start and end-point of the curve segment. This gives well-behaved curves that the controller is able to follow. In the future, the magnitude could be used for better adapting the curve for a certain flight speed. From the end-points and the corresponding derivatives it is easy to derive the following expressions for calculating the coefficients of τ.

$$\mathbf{a}_0 = \tau(0) \tag{7.4}$$
$$\mathbf{a}_1 = \dot{\tau}(0) \tag{7.5}$$
$$\mathbf{a}_2 = 3(\tau(1) - \tau(0)) - 2\dot{\tau}(0) - \dot{\tau}(1) \tag{7.6}$$
$$\mathbf{a}_3 = 2(\tau(0) - \tau(1)) + \dot{\tau}(0) + \dot{\tau}(1) \tag{7.7}$$

7.2.2 Trajectory-Following Controller

The trajectory-following controller, TF, was developed by Conte, Merz and Duranti [13] for flying paths generated by the path planner. A path produced by the PRM and RRT planners consists of a series of path segments, where each segment is a parameterized curve produced by the local path planner. These segments are fed to the controller, one at a time, during the execution of the plan.

When the controller initiates flight along a new path segment, it calculates the curve, τ, from the endpoints and their derivatives in the manner described in the previous section. This results in the same curve that was earlier generated by the local path planner during the PRM or RRT planning process and it is therefore guaranteed to be collision free.

The different control channels of a helicopter are not independent, so an input on one channel affects the dynamics in several dimensions. To deal with this issue, the TF controller is divided into two layers. An inner loop is used for decoupling the control channels of the helicopter and provides a

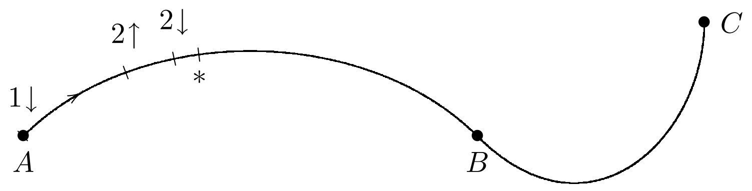


Figure 7.2: Illustration of segment switching when flying along a multi-segmented path.

simpler interface with independent control channels, while the outer loop determines the proper accelerations for these channels in order to track the path. The tracking of the path is performed by a position controller that operates in the plane orthogonal to the path, and the progress along the path is controlled by a velocity controller that has the helicopter flying at the requested speed where possible. For sections of the path with too large curvature for tracking at the requested speed, the controller automatically reduces the speed to stay within the acceleration limits of the helicopter.

That the different constraints are satisfied is only guaranteed as long as the helicopter stays within segments it has received from the path planner. Therefore, it is crucial that the next path segment is available when the helicopter approaches the end-point of a segment. For this reason, the controller requests a new segment when it approaches the point after which it is impossible to stop within the current segment. The interaction with the controller is illustrated in figure 7.2. The calling TP initiates flight along the path by sending the AB segment to the controller at $1\downarrow$. The point after which the helicopter will not be able to stop before B is marked with $*$ in the figure. When the helicopter approaches this point, the controller sends a request for a new path segment to the TP at $2\uparrow$, and the TP responds by sending down the next segment, BC, at $2\downarrow$. If no new segment is available at $*$, the controller will initiate an emergency break at that point, and stop before leaving the AB segment. This interaction between the controller and the calling TP is repeated for each segment of the path. When the last segment is sent to the controller, it will be marked with an end velocity, $v_{\text{end}} = 0$, which indicates that the controller will stop at the endpoint and go into hover mode instead of requesting further segments.

7.2.3 Proportional Navigation

Originally, the path planner was designed to be used in conjunction with a proportional-navigation controller, PN. The PN controller was primarily designed for intercepting moving targets and being able to switch targets rapidly. When used with the path planner, the PN controller is given the waypoints along the path one at a time. The flight along the path is initiated by having the helicopter fly towards the first waypoint and when that waypoint is reached, the controller is commanded to switch to the next one.

The curves traveled with the PN controller have been shown to be approximately of the form of a cubic polynomial curve[3]. However, this approximation is only valid when the angle between the direction of flight and the direction to the next node is small. Thus, the local path planner corresponding to this controller is limited to connect paths to waypoints that are within a 45° angle from the current direction of flight. In section 9.4, we will see that this limitation has different effects on the two methods for handling motion constraints that were described in section 5.2. While the PRM algorithm with nodes in the state space suffers from a reduced efficiency, the multi-level path planning algorithm is hardly affected by this restriction.

A more serious problem with using this controller for flight along paths from the path planner is that it is very sensitive to perturbations, e.g., from wind. This is not very surprising since the PN controller is designed to fly towards a point, rather than following a specified curve. If the helicopter drifts during flight, the PN controller will make no effort to return to any particular flight path, but instead finds a new best trajectory from the current point. This behavior is problematic when flying in an environment with obstacles since only the path received from the path planner is guaranteed to be collision free. It can also be compared with

[3]The curve form differs from the one in section 7.2.1 in that parameterization is linear along a vector from the start to the goal point, and that it is only 2-dimensional in the horizontal plane; the altitude control is handled separately.

the operation of the TF controller, which actively attempts to stay on the specified path.

A more thorough discussion of the use of the path planner in conjunction with the PN controller is given in [43].

7.3 Plan Execution

We have now looked at all the different components involved in flight with the path planner, and we will turn to how they interact to make the helicopter move from one location to another. In figure 7.3, we see the components used for flying to a point and how the information flows between them. The process is initiated by a call to the NavToPoint TP, which calls the path planner with the current position and the goal position, together with any additional constraints. The path planner produces a path, which is sent to the Fly3D TP. Fly3D handles the flight along the path by sending the path segments to the controller in the manner that was described in section 7.2.2. It is also responsible for handling hover points in the path that may occur when using multi-level nonholonomic constraint handling. This includes sending an appropriate yaw command at the end of the first path segment after having achieved stable hover, and initiating flight along the next segment when the yaw motion is completed.

As a more complex example, we will consider the following task: We have an area with a number of buildings, and we would like to have a picture of each of the facades of the buildings. This type of scenario has several realistic applications. The pictures could for instance be used as textures for creating a 3D model of an environment for use in a virtual reality system. In such a scenario, the use of a UAV could be required to get good camera angles for difficult parts of the building that is perhaps not reachable without flying, e.g. roofs. Another similar application could be visual safety inspections of structures such as bridges or large factory plants, where many areas can be hard to reach on foot.

The different components of the system involved in this type of mission are shown in figure 7.4. In this case, the execution is initiated in the Photogrammetry TP, which is responsible for taking a picture of each

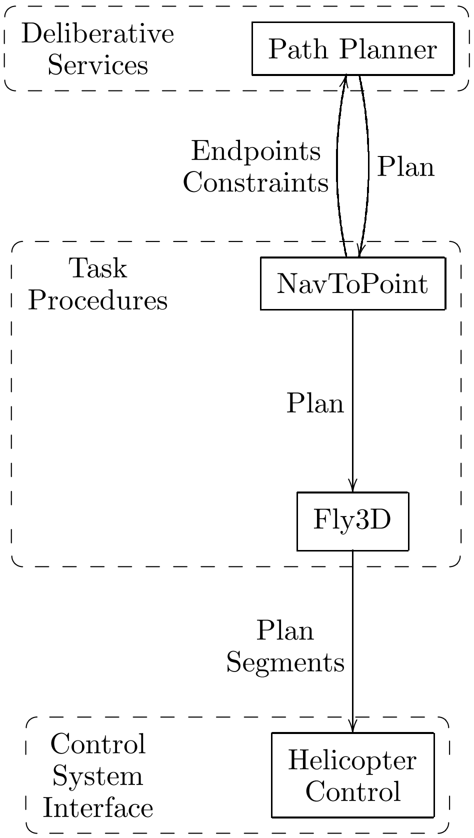


Figure 7.3: The major system components for flight to a point.

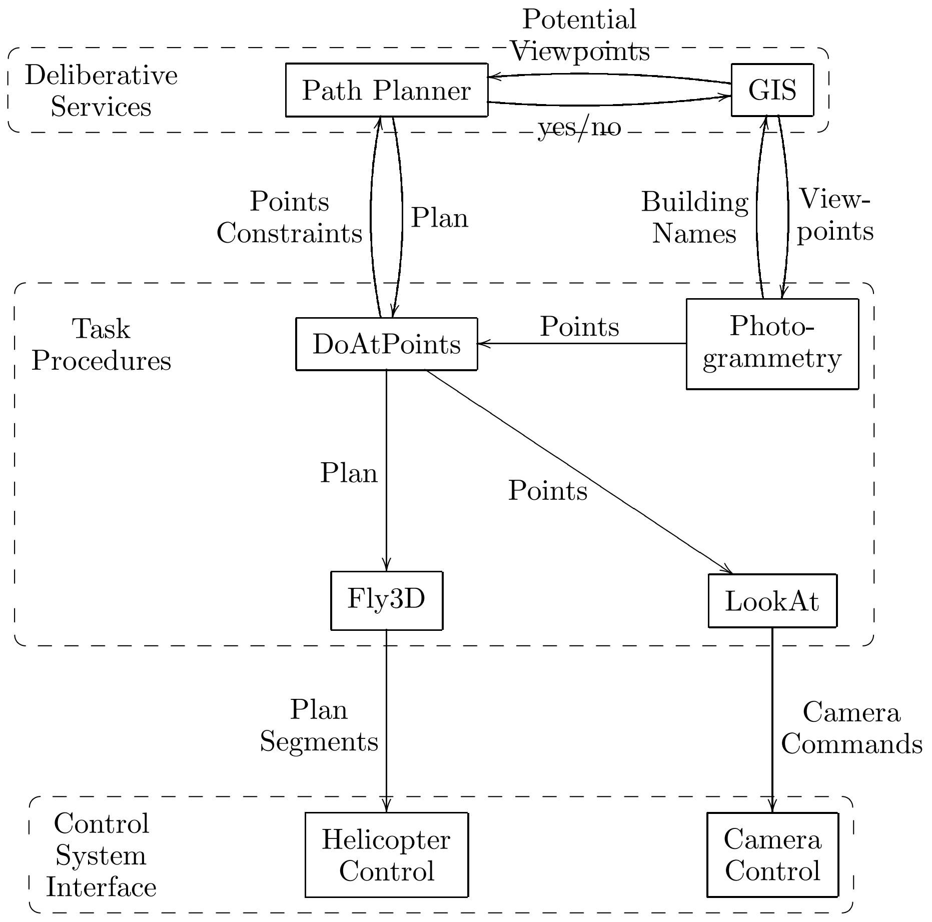


Figure 7.4: The major system components for photographing buildings

facade on a number of buildings. To perform this task it uses the onboard GIS service to locate appropriate camera viewpoints for each facade. The viewpoints are found by testing points in a conical search pattern, starting with the optimal viewpoint in front of the facade, and continuing outwards with points of different distance, elevation, and azimuth from the center of the facade. For each point a visibility test is performed by the GIS service to ensure that the complete facade is visible. The helicopter must also be able to reach the position, which is determined by a query to the path planner. The path planner resolves the query by checking that the point does not fall within the safety margin around obstacles or violate any other path planning constraint. The first point that satisfies both the visibility requirement and all path planner constraints, is used for taking a picture of the facade.

The flight between the points is managed by the DoAtPoints TP, which is a general TP for performing a task on a set of locations. In this case, DoAtPoints is called upon for flying to each viewpoint and taking a picture of the corresponding facade. The points are reordered in an optimal sequence by the path planner, and a plan is generated from the sequence of points. Each sub-path between two viewpoints is executed with the Fly3D TP in the same manner as in the previous example. At each viewpoint, DoAtPoints starts an instance of the LookAt TP, which points the camera to the facade corresponding to the current viewpoint. When the LookAt TP reports that the camera has locked on the requested point and a picture has been captured, DoAtPoints can continue with the flight to the next viewpoint.

Part III

Chapter 8

Flight Tests with the Helicopter

An important goal of this thesis has been to show how a path planner can be developed for practical use with an autonomous helicopter. The integration of the path planning framework developed in chapter 5 with the WITAS UAV platform was described in the previous chapter. In this chapter, an account is given of some of the flight tests that have been performed successfully with the PRM planner. The examples are chosen to illuminate the different capabilities of the planner and how a path planner can be used onboard an autonomous helicopter for real world missions.

8.1 Interactive Camera Positioning

A robot equipped with a path planner can be used for operation at different levels of autonomy. In this first mission, we will see how the path planner can be used in an interactive UAV system, where the operator is mainly interested in getting images from a number of locations. For such applications, the paths taken by the helicopter are of minor interest to the operator, whose main interest is positioning the helicopter at good viewpoints. With a path planner integrated in the system, the operator is freed from the tedious task of specifying each intermediary waypoint in or-

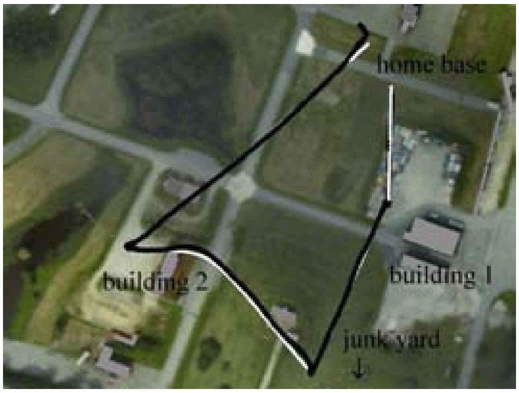
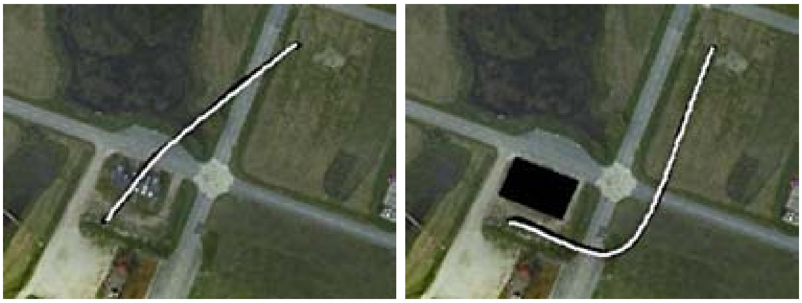


Figure 8.1: An interactive mission visiting a number of locations in the flight-test area in Revinge. The planned paths are white and the logged data from the flight is black.

der to produce safe paths for the helicopter between the points of interest. Instead, the operator only needs to point to the appropriate location on a map, and the UAV can find suitable paths by itself. This greatly improves the ease of use of the UAV system for these types of missions.

Figure 8.1 depicts a series of paths from such an interactive mission. Both the planned paths (white) and the logged data from the flight (black) are shown. The UAV starts from home base and flies to building 1, where it is instructed to point the camera at the opening of a garage located in the building. After having captured a video sequence of the garage, the ground operator decides to continue the flight to a junk yard to gather additional data. Before returning to home base, the operator commands the UAV to fly to a point on the other side of a nearby building. In all these cases, the helicopter dynamically finds a collision-free flight path, which is shown to the operator while the helicopter is hovering autonomously. The helicopter then waits for an acknowledgement from the operator, before it initiates the flight along the path.



(a) Without no-fly zone (b) With no-fly zone (black rectangle)

Figure 8.2: Autonomous flight between two points where the path planner first suggests a path over a building, which the operator then explicitly forbids by introducing a no-fly zone (black rectangle). The plans are shown in white overlapping the logged flight paths in black.

8.2 Use of Runtime Constraints

Runtime constraints were introduced in section 5.3 as a way of improving the flexibility of the PRM planner. A flight test that demonstrates this capability is shown in figure 8.2, where an additional constraint is imposed with the helicopter already flying. In this flight test, the helicopter is asked to fly from a point in the upper right of figure 8.2(a), down behind a house in the middle of the picture. The plan (white) is displayed to the operator, and after approval, the helicopter flies over the house to the goal position (logged flight-path shown in black). The operator might disapprove of the path above the house for various reasons, and can in such cases mark a no-fly zone covering the building (figure 8.2(b)). The no-fly zone is handled by excluding edges that violate this constraint during the graph search in the way described in section 5.3. The resulting flight path is shown in figure 8.2(b).

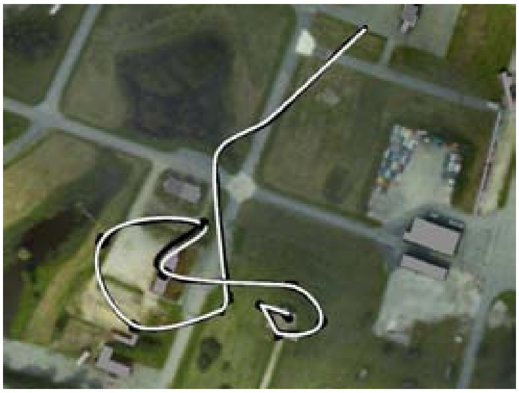


Figure 8.3: Autonomous survey mission to two buildings, which are photographed from each side.

This functionality has also proved useful in practice during the flight tests with the helicopter. One example of this is that we for safety reasons want to avoid having the helicopter flying over the truck containing the ground station. This can be accomplished by putting a no-fly zone over the truck.

8.3 Photogrammetry

A third flight test that has been performed with the helicopter is a mission where the helicopter captures a series of images of each facade from a number of buildings.

In contrast to the two previous flight tests, this mission is performed fully autonomously by the helicopter, without human intervention after the initiation of the flight. The planned path (white) and the path flown by the helicopter (black) are shown in figure 8.3. The helicopter again starts from a hover over the home base and a number of buildings are selected by the operator. For each of the facades of the selected buildings, the GIS service finds suitable positions and camera angles for taking the pictures, and the path planner generates a path that visits all of the viewpoints. The plan is displayed for the operator, and after confirmation, the plan is executed by the helicopter. The helicopter flies from viewpoint to viewpoint, stopping at each one of them, and takes a picture when the helicopter and camera is aligned properly.

Chapter 9

Comparisons between Path Planning Algorithms

This chapter contains a series of empirical investigations of the performance of the PRM and RRT algorithms together with the extensions and adaptations of the algorithms that were presented in chapter 5. The evaluation focuses on performance issues such as completeness, speed and the length of planned paths.

First, the methodology that has been used for the tests is presented, and then a comparison follows on how the PRM and RRT algorithms perform according to various measures. The next two sections discuss the impact of runtime constraints and the efficiency of the two methods for handling motion constraints that were described in chapter 5.

9.1 Method

9.1.1 World models

As described in section 6.1, the main flight-test area used in the WITAS project is a training area for rescue workers in Revinge. From the 3D model representing this area, two different test environments have been derived in order to investigate how the planning algorithms perform under

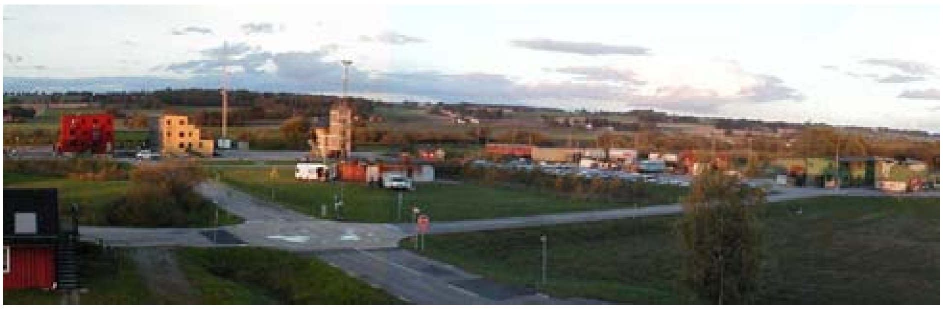


Figure 9.1: The primary flight test area in Revinge.

different conditions.

The first of the environments consists of the eastern part of the training field, and is the location of all the flight tests done in Revinge. This area is relatively open, and a number of buildings, trees, masts, and lampposts are the major obstacles (see figure 9.1). This test environment will be referred to as the **open area**.

This first area consists mostly of open areas, and is therefore less challenging for modern path planning algorithms. Many applications of future UAVs will be performed in complex environments such as urban areas with many large buildings and other obstacles. This makes it interesting to see how the planners perform under such circumstances. For these tests, such an environment was emulated by a geometric scaling of the western part of the Revinge training facilities. This area contains a large number of low, one-story buildings separated by narrow roads. The buildings (together with other objects) were scaled 20 times vertically to produce the city-like environment shown in figure 9.2. This environment will be called the **cityscape**. For this environment the normal 8 meter[1] padding of the obstacles was reduced to 4 meters in order to make it possible to find paths between buildings. This has an effect similar to a uniform scaling of the environment by two, which was done to produce an interesting environ-

[1]The padding consists of a 6 meter safety margin and the 2 meter radius of the helicopter as explained in chapter 6.1.

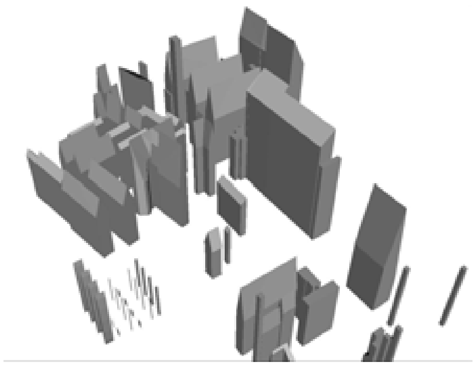


Figure 9.2: The scaled houses forming the cityscape environment.

ment where good planners can find paths between buildings while worse planners cannot.

9.1.2 Implementations

The performance of the different path planning algorithms is influenced by many aspects of the particular implementations. For that reason, an account of some of the choices made in the implementations is given in this section, together with some arguments for ruling out certain sources of error in the comparison. However, large parts of the implementation of the different planners are identical and the results are mostly based on relative performance. Thus implementational details have only a minor influence on most of the results in the following sections.

PRM: Roadmap Generation

The roadmap-construction algorithm (algorithm 4.1.1) is implemented in Java and is adapted for a directed graph, as described in section 5.2.1, for use with the local planner of the helicopter.

The nodes are generated randomly with a uniform distribution within the areas described above. Each added node is connected to the 30 first other nodes found that are within 50 m and to which a connection can be made. The roadmap graph is represented as a list[2] of nodes and a list of edges, where each node keeps a list of out-bound edges, and each edge holds a start and end node.

PRM: Runtime Planner

The PRM runtime planner is also implemented in Java. The graph search is performed with the A* algorithm and is implemented with the heap data structure from the XXL-library[3].

RRT planner

The RRT planner is implemented in Scheme and is compiled to Java byte code with Kawa[4]. Only the higher levels of the algorithm are implemented in Scheme while the lower-level data types e.g. vectors etc. and the collision checker are the same Java implementations that are used with the PRM planner. The choice of programming language has only a small influence on the time measurements since most of the time is spent doing collision checking. Profiling of the program shows that the time spent in collision checking for the RRT algorithm is around 70–75%. For other measurements the choice of programming language has no influence whatsoever.

For the tests with the RRT planner, the expansion step was set to $d = 30$ m, and the search was terminated after 500 iterations.

Collision Checker

The collision checker used for the tests is a Java implementation of the OBBTree algorithm described in section 6.2.

[2] All Java lists are of the type `java.util.ArrayList`

[3] `http://www.xxl-library.de`

[4] `http://www.gnu.org/software/kawa`

9.1.3 Test Setup

All tests, except where explicitly noted, were made by generating 500 collision-free points in each of the two environments. These points were used directly for the coverage measure, and used pair-wise for the other tests of the planners, giving 250 test instances for success rate, planning time, and path length measurements. The same points were used for all different planner variations in each environment.

Each test case is specified by a number of parameters:

Planner The type of planner: PRM or RRT.

Model The environment in which the tests were run: open area or cityscape.

Controller The type of controller for which the plan is generated: either the trajectory-following controller, TF, or the proportional-navigation controller, PN. The TF controller has been used in all of the following tests, except where explicitly noted.

Motion constraint approach The method for handling motion constraints: either the state-space approach or the multi-level approach. The state-space approach has been used in all of the following tests, except where explicitly noted.

Nodes The number of nodes in the roadmap for a PRM planner.

For each of the test cases the following measurements were made:

Success Rate The percentage of pairs of points for which the planner succeeded in finding a solution.

Planning Time The time for planning in the online phase. This was measured taking the value of the system time before and after the call to the planner. To equalize for the effect of the garbage collector, the Java virtual machine was started with an incremental garbage collector.

Path Length The path length was measured numerically by approximating each curve segment of the path by 10 linear segments. This measurement was taken after applying the smoothing operations and curve replacements.

Roadmap Creation Time The time to create the roadmap.

Roadmap Coverage The percentage of the configuration space that is directly reachable from the roadmap by one application of the local planner. The coverage was measured by taking 500 free points and count the percentage of these that can be connected to the roadmap.

When comparing the planning time and path lengths for different planners, only plans where all planners in the specific comparison succeeded are included. This is done to remove the effect of different success rates for different planners in cases where the planners perform differently for success and failure. The failure behavior is analyzed separately.

All the tests were done on a PC with an AMD Athlon XP 1800 processor and 512 MB of memory, running Linux.

9.2 Comparisons between PRM and RRT

This section compares the performance of the PRM planner with different roadmap sizes and the RRT planner. In all these tests the local planner corresponding to the TF controller is used and the motion constraints are handled with the state-space approach. The choice of local planner and the method for handling motion constraints have little importance for the comparisons in this section and are discussed separately in section 9.4.

9.2.1 Completeness

In table 9.1, we see that the success rate of the PRM planner gradually increases when the number of nodes goes up, before it saturates at a high level. In the open area, already a small roadmap provides enough coverage for successful planning with the PRM planner. The RRT planner also performs well with a 100% success rate.

planner	nodes	open area	cityscape	edges in cityscape
PRM	100	25%	17%	241
PRM	250	93%	71%	1808
PRM	500	100%	83%	6674
PRM	1000	100%	84%	26435
PRM	2000	100%	87%	76898
PRM	3000	100%	88%	130091
RRT	-	100%	52%	-

Table 9.1: Comparison of the success rates of a PRM based planner with some different size of roadmaps and an RRT planner in the different test environments.

In the cityscape, the performance is worse, but the PRM planner still performs well given a sufficiently large roadmap. The fact that the PRM planner does not come closer to 100% is surprising, given that the coverage of the roadmap was measured to 98.0% for the 3000-node roadmap. This can be attributed to small isolated regions of the environment where collision-free nodes can be located that are unreachable or very hard to reach from other parts of the environment. Such nodes contribute to the coverage since they can be connected to nodes sampled in these regions, but the planner still fails to find plans to these regions since they are not connected to the rest of the roadmap. In an environment of this difficulty, the RRT planner is too weak for practical use with a success rate of only 52%.

One further interesting point can be surmised from the table. Comparing the number of nodes used by the RRT planner with the different roadmaps used with the PRM planner, we see that the RRT planner, using a cut-off value of 500, performs much worse than the PRM planner with a 500-node roadmap. However, the tree constructed by the RRT planner only uses a maximum of 500 edges which falls somewhere between the 100 and 250 node roadmaps, and closer to the 100 node roadmap. This indicates that the RRT planner makes good use of the knowledge of start and goal point when constructing the trees, and that it does have a more efficient method for covering the environment.

planner	nodes	roadmap-time (s)	online-time (ms)
PRM	100	1.00	*
PRM	250	6.50	52
PRM	500	21.89	108
PRM	1000	75.64	193
PRM	2000	249.96	407
PRM	3000	532.28	601
RRT	-	-	660

Table 9.2: Comparison of the mean planning time in the cityscape for the PRM and RRT planners, and the time for constructing the roadmaps for the PRM planner. Only planners with more than a 50 % success-rate are included in the comparison on planning times, and only plans where all these planners succeeded are included in the numbers above. Planners that failed in more than half the cases are marked with * above.

9.2.2 Planning Time

The mean planning times in the cityscape are shown in table 9.2. As can be seen, the PRM planner is faster than the RRT planner, especially with smaller roadmaps, but even for the large 3000-node roadmap, the difference is significant. However, it is important to remember that the RRT planner operates without the use of a precompiled roadmap, which explains the longer planning times, but also makes the RRT planner more flexible in use. It is possible to compile the roadmap for the PRM planner in the online phase and get the same flexibility as the RRT planner. However, for such a planner we see that the roadmap construction time together with the online planning time is much longer than the planning time for the RRT planner, if we want to ensure the same success rate (more than 250 nodes according to table 9.1).

The variance of the planning times is also of interest, and is shown in figure 9.3. The difference between the collected distributions of the PRM planner and the wider distribution of the RRT planner is striking, with the RRT planner often being faster than the PRM planners, even if the mean planning time was shown to be longer above. The reason for this

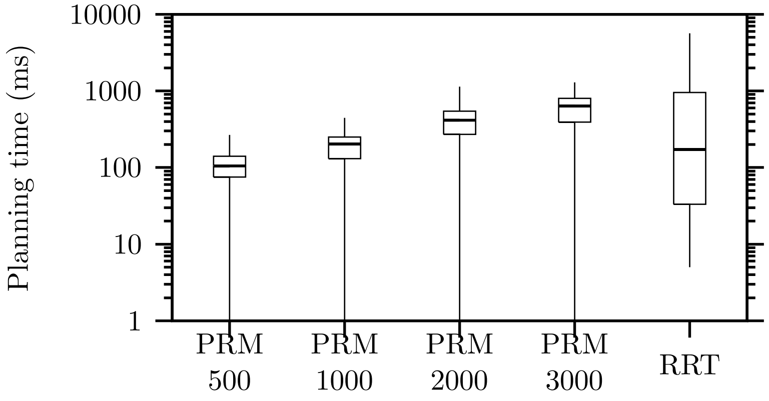


Figure 9.3: Comparison of the planning times in the cityscape environment for the PRM planner with various roadmap sizes and an RRT planner. Only planning times for plans where all planners succeeded are included.

discrepancy is of course the poor worst-case performance, which is more than ten times slower than the PRM planner with a 500-node roadmap.

In the open area, the situation is quite different as can be seen in figure 9.4. Here the RRT planner performs much better and is comparable in planning time to the PRM planner with a 250-node roadmap. Even though the variance of the RRT planner is still larger[5] than for the PRM planner, it is much more acceptable in this, simpler environment.

For a better view of the planning time distributions for the PRM and RRT planners, histograms over the distributions are shown in figure 9.5 and 9.6. Comparing the cases where the planners succeed (full outline), we see some differences between the planners. The RRT planner is very fast in many instances; many of the plans in the leftmost box are actually done under 100 ms which cannot be seen in the diagram due to the low resolution, but for other cases the RRT planner may need several seconds to find a plan. The PRM planner, on the other hand, has a more collected distribution.

[5]The variance downwards for the PRM planners, towards shorter times cannot be considered a drawback.

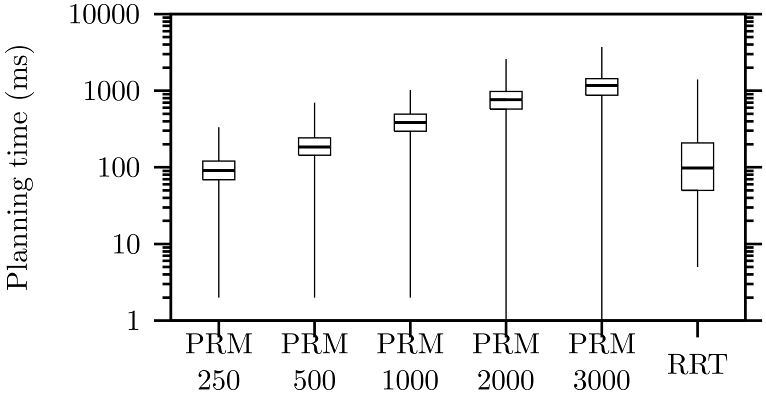


Figure 9.4: Comparison of planning times for the PRM planner with various roadmaps and a RRT planner in the open area. Only planning times for plans where all planners succeeded are included.

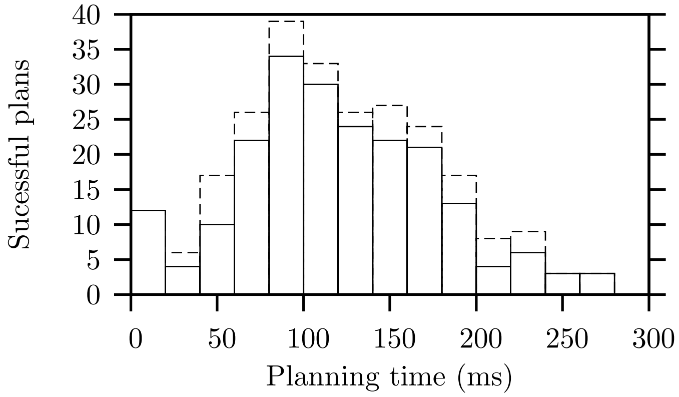


Figure 9.5: Histogram showing the number of successful plans (of totally 250 plans) taking different amounts of planning time for the PRM planner with a 500-node roadmap in the cityscape. The solid-line boxes show successful plans, while the dashed-line boxes show failed plans.

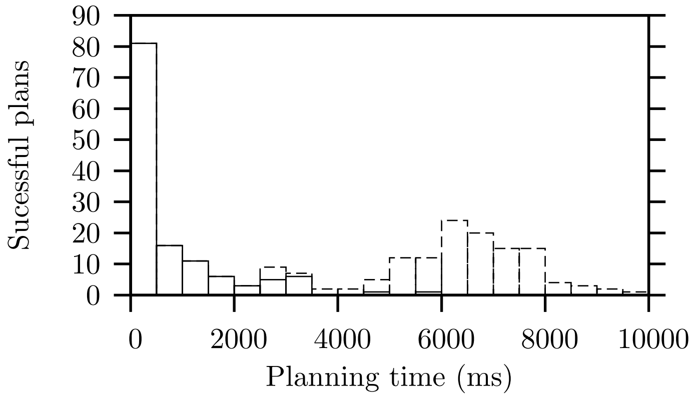


Figure 9.6: Histogram showing the number of successful plans (of totally 250 plans) taking different amounts of planning time for the RRT planner in the cityscape. The solid-line boxes show successful plans, while the dashed-line boxes show failed plans.

Looking at the cases where the planners fail (dashed lines in diagram 9.5 and 9.6) an interesting pattern emerges. While the PRM planner's failure distribution follows the distribution of the successful plans, it takes much more time for the RRT planner to report a failure. This is due to the difference in the basic layout of the algorithms. The PRM algorithm can fail both in the initial connection of the starting point to the roadmap, and when trying to connect nodes reached during graph search to the goal configuration. The RRT planner on the other hand fails only if it is unable to connect the two trees, something that only happens when the maximum number of iterations is reached and the search is terminated. The maximum planning time before failure for the RRT algorithm can be brought down by decreasing the cut-off value. However, there is a trade-off between low maximum planning time and high success rate, since a lower cut-off value means that the planner will fail also in some cases where a plan could be found with a higher cut-off value.

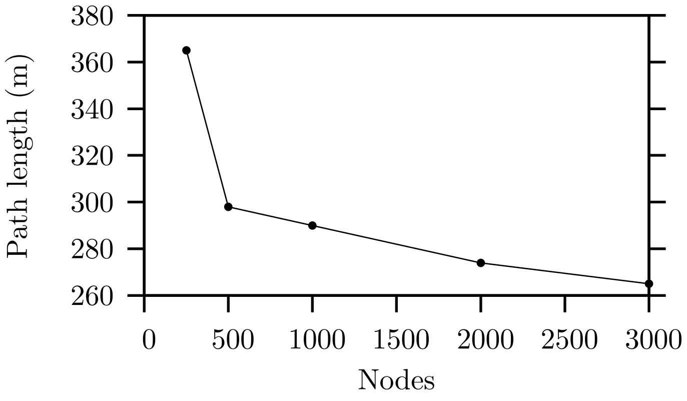


Figure 9.7: The mean length of the solution paths from the PRM planner with different roadmaps in the cityscape (after smoothing).

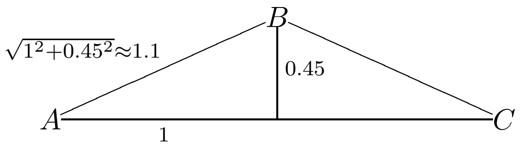


Figure 9.8: The path from A to B to C is approximately 10% longer than the straight line path directly from A to C.

9.2.3 Path Length

Figure 9.7 shows how the mean length of the solution paths from the PRM planner depends on the roadmap size. As can be seen the mean path length decreases as the number of nodes in the roadmap goes up. A decrease from around 290 m for the 500 node roadmap to 265 m for the 3000 node roadmap may not seem significant, but unlike the planning time, where 10% extra time would hardly be noticed by the operator, a 10% longer path implies a large detour as can be seen in the simplified example in figure 9.8. The difference is of course even larger for the 250-node roadmap. This indicates that it is often useful to increase the density of the roadmap, to achieve higher quality solutions, even if the success rate

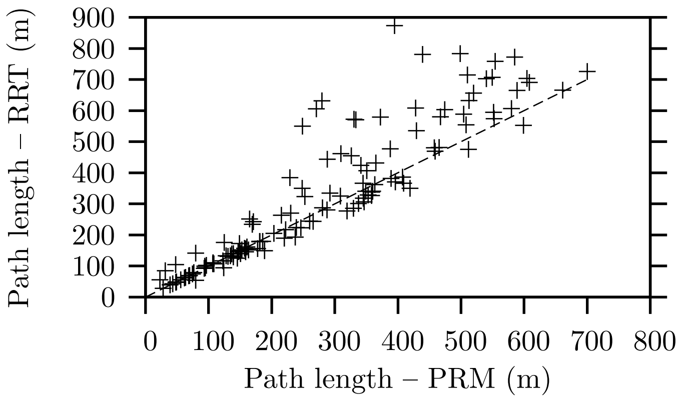


Figure 9.9: A scatter plot relating path lengths for an RRT planner against a PRM planner with 2000 nodes in the cityscape.

does not increase further.

The mean path length for the RRT planner at the same set of problems is 313 m, which is worse than the PRM planner in all cases except for the small 250-nodes roadmap. This is especially true in comparison to the PRM planner with large roadmaps with 2000 or 3000 nodes. The main reason for the difference is not so much that the RRT planner is consistently worse than the PRM planner, but rather that it is occasionally much worse. This can be seen more easily in the scatter plot in figure 9.9 where the path lengths from the RRT planner are related to the path lengths from the PRM planner with a 2000 node roadmap. This diagram shows that the two planners produce paths of approximately the same lengths in most instances, and it is just in a small number of cases that the RRT planner performs worse. However, the differences for these few, longer plans are very large, from around 50 % up to 100 % of the length of the corresponding paths from the PRM planner. As was the case with the planning time, we again see that even if the RRT planner often shows good performance, it is less consistent.

Also in the open area, an increased number of nodes in the roadmap results in shorter path lengths which is shown in figure 9.10. Just as with

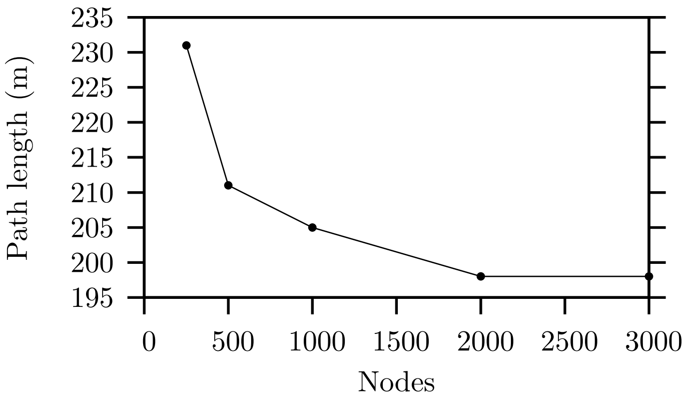


Figure 9.10: The mean length of the solution paths from the PRM planner with different roadmaps in the open area (after smoothing).

the planning time, the RRT planner performs relatively better in the open area, with a mean path length of 212 m, which is better than the smaller roadmaps and just slightly worse than the larger ones.

Looking at figure 9.11, we see that the problem of occasional bad plans remains in this environment, but that it occurs with less frequency.

9.2.4 Discussion

The choice between the PRM and RRT algorithms is largely determined by the complexity of the environment and to what degree the environment and other constraints are known in advance. The PRM algorithm performs better than the RRT algorithm in most situations if equipped with a suitable precompiled roadmap. Thus the preparatory phase, where the roadmap is generated, does pay off if the environment is known in advance. However, the difference between the algorithms is not that large in all cases so other factors may also be of importance.

For the open area, both the PRM and the RRT planners perform well, and even if the PRM planner is slightly better, the RRT planner can probably be considered good enough for practical use in this type of environment.

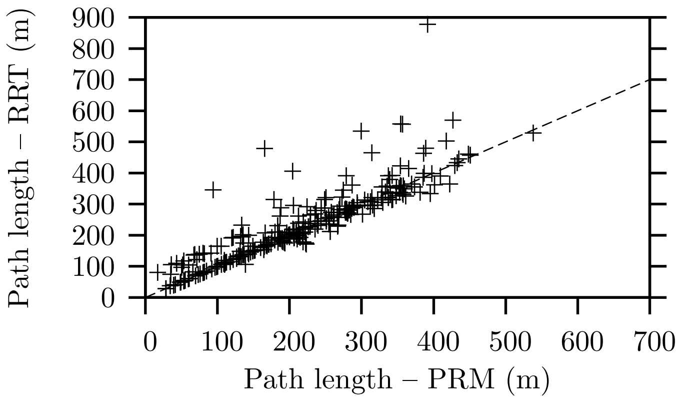


Figure 9.11: A scatter plot relating path lengths for an RRT planner against a PRM planner with 2000 nodes in the open area.

The RRT planner might therefore be the better choice since it does not require any precompiled data structures, which gives it a higher degree of flexibility. The only problematic thing with the RRT planner in this environment is that it occasionally generates bad plans. For some types of robots, execution time is not so critical and it is not a problem if the robot takes a small detour from time to time, but for UAVs, it is generally unacceptable since flight time is often expensive. It may be possible to reduce this problem by tuning the RRT algorithm, e.g., by making more than one call to the planner and taking the best solution.

In the more complex cityscape, the RRT planner fails to find a plan in many cases, while the PRM planner performs well with sufficiently large roadmaps. In this type of environment, some type of precompiled data structures may be necessary to solve the problem efficiently and the PRM planner would probably be a better choice.

If the planner needs to operate under real-time constraints, it is important that the planning time is predictable. In the comparison on planning times in section 9.2.2, we saw that the PRM planner has a more collected distribution of planning times, which is an advantage under such constraints. The RRT planner, on the other hand, has a somewhat unstable

	Without constraint	With constraint	Increase
Unaffected plans	216 ms	240 ms	11 %
Affected plans	275 ms	369 ms	34 %

Table 9.3: Planning times with and without runtime constraints.

performance. In many cases it finds relatively good plans in a short time, but from time to time, the planner takes much longer to find a plan, if it finds one at all. The most natural adaption of the RRT planner for use in a real-time system is to limit the tree-expansion in time rather than in number of iterations. This would lead to occasional failures but the same is true for any heuristic algorithm, including the PRM algorithm. Thus, there must always exist a backup plan that can be used if no plan is found, e.g., an escape trajectory that brings the UAV to a safe state.

9.3 PRM with Runtime Constraints

The addition of runtime constraints to the PRM planner was made in order to increase the flexibility of the planner for real world applications. For this extension to be useful in practice, it is necessary that the addition of runtime constraints does not produce a large overhead on the runtime planning process. In this section we will investigate the magnitude of this overhead. We will also see to what extent it is possible to repair the roadmap if the runtime constraints break the roadmap connectivity.

9.3.1 Timing Tests

In order to investigate the overhead involved in testing constraints in the runtime phase of the PRM planner, a test was made with a no-fly zone covering a 100 m $\times$ 100 m area in the middle of the open area. A roadmap with 500 nodes was used and the test was made with the TF controller. The results of the tests were divided into two categories depending on if the plans were affected by the no-fly zone or not and the results are shown in table 9.3. We see that the addition of the no-fly zone gives a negligible increase in planning cost for plans outside the area occupied

		Nodes	
		250	2000
	2	77 → 95	96 → 96
Zones	10	79 → 96	97 → 97
	50	11 → 37	41 → 50

Table 9.4: The improved success rate with RRT reconnection, with different number of no-fly zones and roadmaps of different sizes. The numbers to the left of the arrows indicate the success rate for the standard PRM algorithm, while the number to the right is the success rate with RRT reconnection.

by the no-fly zone. For plans that are affected by the no-fly zone, the difference in planning time remains small. This shows that this method of adding runtime constraints to the PRM algorithm is viable for increasing the flexibility of the planner in cases where the extra constraints can be evaluated quickly.

9.3.2 Repairing Broken Roadmap Connectivity

Adding constraints to a path planning problem may break the connectivity of the roadmap. To handle this, an optional reconnection attempt was introduced in section 5.3.3 for the query phase of the PRM planner. This extra procedure is called in the case of failure of the graph search and attempts to connect the disconnected graph components with the RRT-Connect planner.

The improvement induced by this RRT-connection step was tested by adding randomly located no-fly zones in the open area. The no-fly zones were constructed by picking a random location, p, within the flight-test area and sampling three random vertices within a 100 m $\times$ 100 m square centered at p. Each set of three vertices was then used to form a triangular no-fly zone. The test was performed with 2, 10 and 50 no-fly zones and roadmaps of 250 and 2000 nodes, and the PRM planner was tested with and without the RRT-connection step on 100 pairs of configurations.

The results of the tests are shown in table 9.4, where the number of successes with the standard PRM algorithm is shown to the left of the arrow, and the number of successes with the additional RRT-connection step is shown to the right.

The table shows that for large roadmaps, a small number of no-fly zones have no effect on the connectivity of the roadmap and the RRT-connection step is never needed. For small, sparse roadmaps, on the other hand, the extra RRT-connection attempt is very helpful, and we get almost the same success rate as with the large 2000 node roadmap. However, it should be noted that a large roadmap is also preferable for improving the path quality, as was shown earlier in this chapter.

For difficult problems, the connectivity starts to break down also for the larger roadmap, and the RRT-connection attempt has a positive effect. The planner performs significantly better with the larger roadmap also in this case, which indicates that the remaining parts of the larger roadmap are still of good use.

9.4 Approaches for Handling Motion Constraints

In chapter 5, two ways were described for handling nonholonomic motion constraints within the PRM method: the state-space approach and the multi-level approach. In this section, it is shown that the multi-level approach is more efficient under certain circumstances, specifically when used with the more restricted proportional navigation controller (PN).

We will focus on the success rate and the planning time for which there are significant differences; the lengths of the plans are approximately the same for the two methods. The main drawback of the multi-level approach when used with an autonomous helicopter is that sharp corners may remain in the final path, where the helicopter must stop into hover before it continues in the direction of the following segment. In the last subsection, we will look at how often this happens.

9.4.1 Roadmap Size

With the multi-level approach, straight-line edges are used throughout the planning process, except for the final smoothing and curve replacement

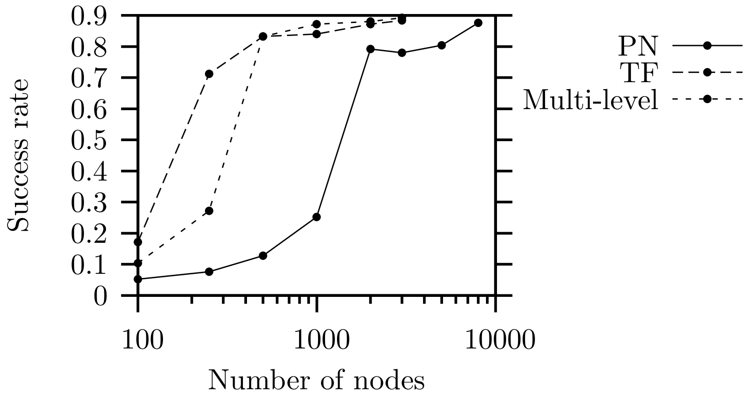


Figure 9.12: Success rate for roadmaps of different size in the cityscape. The graph is similar for the open area.

step. This means that the success rate of the planner is independent of the curve form of the final path. The success rate with the straight-line roadmap of the multi-level approach is shown in figure 9.12 together with the success rate for roadmaps constructed for the TF and PN controllers. As can be seen in the diagram, a much larger number of nodes is required to achieve a good success rate when the curves for the PN controller is used in the roadmap. This is a result of the lower probability of connecting two random nodes under the restrictions of this controller. The roadmaps with the general cubic curves used for the TF controller and the roadmaps with linear paths both reach an acceptable performance at approximately 500 nodes.

Due to the increased difficulty of connecting nodes with the PN curve, the corresponding roadmaps have fewer edges for a certain number of nodes compared to the other two types of curves. Thus, the number of nodes is not a good measure for roadmap size when comparing PN roadmaps with roadmaps generated with the other two curve types. Instead, the measure must incorporate the memory usage for both nodes and edges. Each node contains the three position coordinates for the helicopter configuration which requires 12 bytes if floats are used. For roadmaps in the state

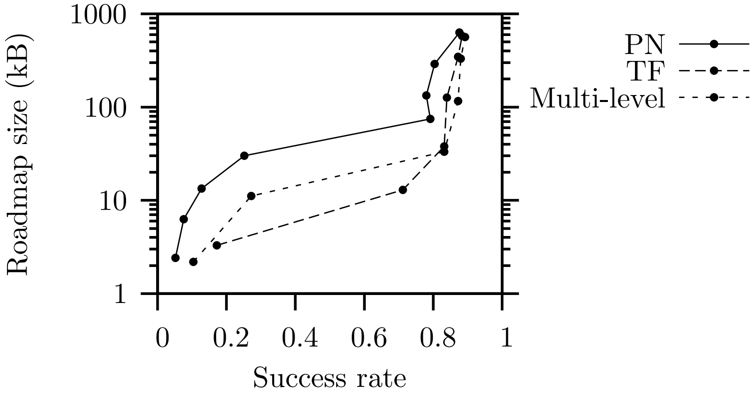


Figure 9.13: Roadmap size required to reach a certain success rate in the cityscape. Each point represent the size and success rate for a roadmap with a certain number of nodes.

space, an additional 12 bytes are required giving the node a total size of 24 bytes. Outgoing edges from a node can be represented as a sequence of pointers to their destinations, which requires 4 bytes per edge. The real implementation of the graph data structure requires more memory due to software abstractions, but this measure can still serve as a reasonable approximation for the relative memory consumptions of the different roadmaps. The sizes of the roadmaps required to reach a certain success rate for the different local path planners is depicted in figure 9.13. We see here that the roadmap with local paths matching the PN controller will take approximately twice as much memory as the other two roadmaps.

9.4.2 Planning Time

The planning times for the two approaches also differ significantly when generating paths for the PN controller. In figure 9.14, we see that the state-space approach requires a planning time that is around thrice that of the multi-level algorithm, to achieve the same success-rate. With the TF controller, the situation is quite different, as can be seen in figure 9.15. Here the difference between the state-space and the multi-level method is

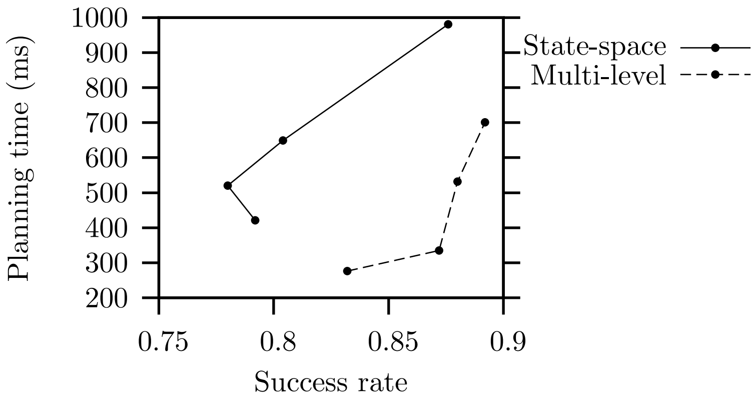


Figure 9.14: Illustration of the required planning time for achieving a certain success rate with the PN controller and the two methods for handling motion constraints. Each point represents the size and success rate for a roadmap with a certain number of nodes. The results are from the cityscape environment.

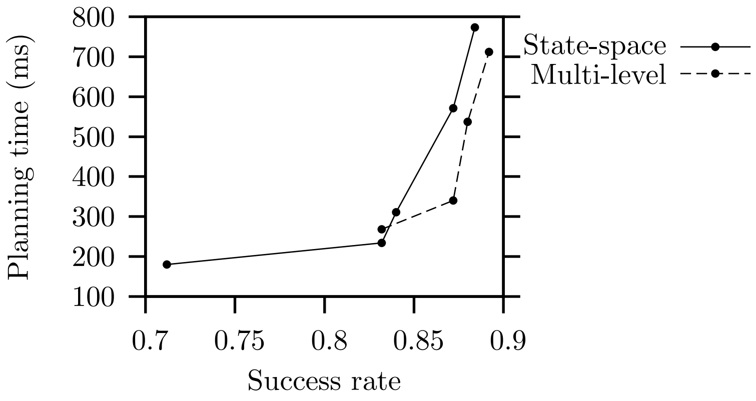


Figure 9.15: Planning times required for different success rates for the TF controller in the cityscape environment.

Model	Curve	0	1	2	>2
open area	PN	96.4 %	2.0 %	1.6 %	0.0 %
open area	TF	96.4 %	2.4 %	1.2 %	0.0 %
cityscape	PN	75.2 %	5.2 %	7.2 %	0.4 %
cityscape	TF	77.2 %	2.8 %	7.2 %	0.8 %

Table 9.5: Occurrences of sharp corners in the final plan with the multi-level approach to nonholonomic constraints.

smaller, but the state-space method is still significantly slower when high success rates are required.

9.4.3 Remaining Sharp Corners

The main drawback of the multi-level method is that the transformation from a piecewise linear to a piecewise cubic path is not always completely successful. If the transformation fails for two neighbouring edges, a sharp corner remains in the path where the helicopter needs to stop and hover, something that is clearly inefficient. In table 9.5 the percentages of paths containing such sharp corners are presented. We see that in the open area, which has been used for flight tests, corners are rare, and occur in less than 5 % of the paths.

In the cityscape, the sharp corners occur more frequently, which is to be expected due the more complex environment with many more obstacles that the helicopter must negotiate. However, for flight in such cramped spaces, occasionally stopping and turning to new flight directions because of nearby obstacles is more acceptable.

9.4.4 Discussion

Delaying the nonholonomic motion constraints until the runtime phase makes it easier to connect nodes during roadmap construction. This makes it possible to achieve good performance with smaller roadmaps also for the more restricted PN controller. The TF controller is more powerful, which makes the difference smaller between the state-space and multi-level approach for that controller. Thus, the multi-level approach is mostly interesting for difficult planning problems, where a weaker local path planner is used.

Besides increased efficiency, the curve-replacement method could also be used to achieve a higher degree of flexibility in the runtime phase, similarly to how runtime obstacle constraints are used. Since the curve replacement occurs at runtime, more information is available for determining the curve form. This means that, e.g., requested velocity can be taken into account for optimizing the curves.

There are however some disadvantages with delaying the motion constraints to the runtime phase. The time to perform the replacement could be a problem if curve generation or collision checking is very expensive. As we saw above, this is not a problem in this domain. Another problem is that the curve replacement is not always successful. For some classes of robots, e.g., locally controllable robots, the replacement is always possible, while for others, e.g., the helicopter, a small number of replacement failures may be acceptable. However, for other types of robots, e.g., fixed-wing airplanes, the transformation to a path that can be flown cannot be guaranteed in general, and any replacement failure means that the path is not traversable. One solution to this problem is to continue the graph search for alternative routes if the curve replacement fails.

Chapter 10

Conclusions

In this thesis, it has been shown how modern path planning algorithms based on random sampling of the configuration space can be used with an autonomous helicopter. In this concluding chapter a summary is given on the major points that have been discussed. In each of these sections, some interesting topics for future research are also presented.

In the first section the integration with the helicopter is discussed. The following section contains a discussion on how constraints can be handled at different stages of the planning process. The last section is on mission planning, a topic that falls somewhat outside the scope of this thesis, but nevertheless is an interesting direction for future research related to path planning.

10.1 Integration with the Helicopter Platform

One of the major goals of this thesis work has been to investigate the integration of a sampling-based path planner in a physical autonomous helicopter. This involves many practical issues such as integration into the system architecture and adaptations of the algorithms to the particular helicopter platform and control system. The solutions to these issues were presented in chapter 7.

Most important of these issues has been to make the path planner produce paths that the helicopter can follow at a reasonable speed. To achieve this goal, it is vital that the flight paths have a low degree of curvature and contain no sharp corners where the helicopter has to stop. Two methods to accomplish this were presented in section 5.2: building the roadmap in the state space and using a multi-level approach. We saw in section 9.4 that the multi-level approach is the more efficient one when a weak local path planner is used and occasional stops are acceptable. With the current, more versatile trajectory-following controller (TF), both methods have comparable performance.

The PRM-based planner that was developed using these methods has been tested on the WITAS UAV on numerous occasions, and three successfully performed experiments are described in chapter 8.

The current implementation of the path planner has proved to produce paths of acceptable quality for the types of missions that have been performed so far with the helicopter. However, there are a number of weaknesses with respect to optimization of the paths that should be addressed in future work. The most important issues to address are improvements of the local path planner for producing curves that are closer to optimal and performing the graph search with other cost functions than path length, e.g., flight time or fuel consumption.

10.2 Changing Constraints

The practical flight tests have shown the importance of a flexible path planner that can adapt to the requirements of specific missions. With single-query planners, e.g., RRT planners, such a flexibility is a natural consequence of that the problem does not need to be specified until the runtime phase. For PRM planners, the situation is more problematic due to the reliance on precompiled roadmaps, which led to the development of the runtime constraints capability described in section 5.3. A flight test demonstrating the use of runtime constraints was described in section 8.2, where the operator added a no-fly zone around a building at runtime. Adding such a no-fly zone was shown to give only a small runtime overhead in section 9.3.

Pruning edges from the graph during search solves only a part of the problem with additional runtime constraints. Even if a certain roadmap is appropriate for path planning in a specific environment, additional runtime constraints may increase the difficulty of the problem and the original roadmap connectivity may be broken. In order to extend the roadmap in such cases, an RRT extension step was added to the PRM planner in section 5.3.3. This addition was shown to produce a significant improvement in success rate for planning with difficult runtime constraints in section 9.3.2.

Expanding the roadmap by using RRTs is also of great use when obstacles disappear and new passages may be formed. If the removal of an obstacle creates a passage between two parts of the configuration space with disconnected roadmap components, the RRT-connection step can be used to expand the roadmap so that a path can be found between those two components.

At this point only a basic version of the RRT expansion is implemented, which attempts to connect two roadmap components by extending trees from two selected nodes. A more sophisticated algorithm, which also would take plan quality into account, could be formulated by first performing a bidirectional A* search from the start and goal configurations, and using these search trees as the initial trees for the RRT algorithm. Such a strategy blends well with the A*-like optimization of the RRT algorithm described in section 4.2.4, and could perhaps be developed to find better plans even in cases where a solution is found in the original roadmap.

Taken together, the runtime constraints and the RRT expansion can be used as the basis for an incremental PRM algorithm. Such an algorithm would start by constructing a possibly sparse or even disconnected roadmap for the environment. For failing queries, the roadmap could be extended by application of the RRT algorithm. Such an algorithm would be robust to changes in the environment since new obstacles invalidate parts of the roadmap and the roadmap can grow in areas where obstacles have been removed. In a very static environment, such an algorithm would reduce to the standard PRM algorithm since solutions to the queries would normally be found by the graph search. In an extremely dynamic environment, on the other hand, it would instead be similar to the RRT algorithm since moving obstacles would invalidate large parts of the roadmap.

10.3 Mission Planning

The building photography flight, described in section 8.3, is an example of a typical mission for a UAV. To perform this mission, the UAV needs to find appropriate viewpoints for each facade of the buildings, order the points to minimize path length, and plan a path connecting the points. Each of these steps is performed fully automatically for any set of buildings, based on information from the GIS-service and the planning capabilities of the path planner. However, the overall mission strategy containing these steps is currently setup manually in order to achieve the goal of having a picture of each facade. It would of course be of great benefit to have a planner that is able to set up these types of missions automatically for such goals, and below an outline is given of how this could possibly be done by extending the capabilities of sampling-based path planners.

The building photography mission is typical for a UAV mission in that it is primarily concerned with observations, which requires moving the UAV appropriately for successful operation of its sensors. This type of problem puts less emphasis on ordering constraints than what is common in many classical planning domains, where the agent performs actions that directly affect the world. A major challenge in planning for UAVs is instead the continuous domains that often include complex constraints over real-valued variables, e.g., obstacles and visibility requirements.

To some degree, this type of problem is what path planners are specifically designed to solve. However, path planning, as defined in chapter 2, is not sufficient for finding complete mission plans, since the only concern is to find a path from one point to another in the robot's configuration space. This kind of problem instead requires other types of actions, e.g., pointing the camera and taking pictures, or switching between different flight modes such as landing, take-off, or hover.

One interesting line of research would be to extend path planners to handle actions of the type normally used with classical planning. The configuration space of the robot could be extended by adding further discrete or continuous state variables, e.g., a binary state-variable describing if the UAV has got a picture of a certain building. As was described in section 7.3, the viewpoints in the building photography mission were found by sampling points in front of each facade. This is similar to the path planning process itself in sampling-based path planners and gives some indications that for these types of problems, methods similar to the PRM and RRT algorithm may be suitable also as components in higher-level planners. They would then be used for finding trajectories through a hybrid continuous/discrete state-space with both geometric dimensions of the configuration space for the robot and discrete dimensions describing discrete events, e.g., taking a picture. By planning in continuous domains, problems such as covering an area with a video sequence would also have natural formulations.

Bibliography

[1] Mert Akinc, Kostas E. Bekris, Brian Y. Chen, Andrew M. Ladd, Erion Plakue, and Lydia E. Kavraki. Probabilistic roadmaps of trees for parallel computation of multiple query roadmaps. In *Proceedings of Eleventh International Symposium on Robotics Research*, October 2003. `http://www.cs.rice.edu/~aladd/pubdir/conference/isrr2003paper.pdf`, accessed: November 19, 2003.

[2] N. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 113–120, 1996. `http://citeseer.ist.psu.edu/amato96randomized.html`.

[3] Jérôme Barraquand, Bruno Langlois, and Jean-Claude Latombe. Robot motion planning with many degrees of freedom and dynamic constraints. In *The fifth international symposium on Robotics research*, pages 435–444, Cambridge, MA, USA, 1990. MIT Press.

[4] Jur P. van den Berg and Dennis Nieuwenhuisen and Léonard Jaillet and Mark H. Overmars. Creating robust roadmaps for motion planning in changing environments. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems - IROS'05*, 2005. `http://www.cs.uu.nl/people/berg/IROS2005-1.pdf`, accessed: Friday, August 5, 2005.

[5] Jur P. van den Berg and Mark H. Overmars. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on*

Robotics, 2005. http://www.cs.uu.nl/people/berg/ITRO.pdf, accessed: Friday, August 5, 2005.

[6] R. Bohlin and L. E. Kavraki. Path planning using lazy prm. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 521–528. IEEE Press, San Fransisco, CA, April 2000. http://www.cs.rice.edu/CS/Robotics/papers/bohlin2000lazy-prm-icra.pdf, accessed: November 24, 2003.

[7] R. Bohlin and L. E. Kavraki. A randomized algorithm for robot path planning based on lazy evaluation. In P. Pardalos, S. Rajasekaran, and J. Rolim, editors, *Handbook on Randomized Computing*, pages 221–249. Kluwer Academic Publishers, 2001. http://www.cs.rice.edu/CS/Robotics/papers/bohlin2001lazy-evaluation.pdf, accessed: November 24, 2003.

[8] Valérie Boor, Mark H. Overmars, and A. Frank van der Stappen. Gaussian sampling for probabilistic roadmap planners. http://citeseer.ist.psu.edu/boor01gaussian.html.

[9] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Proceedings of IROS-2002*, October 2002. http://www.cs.cmu.edu/~mmv/papers/02iros-rrt.pdf, accessed: November 19, 2003.

[10] J. Canny and J.H. Reif. New lower bound techniques for robot motion planning problems. In *28th Annual IEEE Symposium on Foundations of Computer Science*, 1987.

[11] John F. Canny. *The complexity of robot motion planning.* MIT Press, Cambridge, MA, USA, 1988.

[12] P. Cheng, Z. Shen, and S. M. LaValle. Using randomization to find and optimize feasible trajectory for nonlinear systems. In *In Proc. Annual Allerton Conference on Communications, Control, Computing*, pages 926–935, 2000. http://msl.cs.uiuc.edu/rrt/papers/CheSheLav00.ps.gz, accessed: November 19, 2003.

[13] Gianpaolo Conte, Simone Duranti, and Torsten Merz. Dynamic 3d path following for an autonomous helicopter. In *Proceeding of the 5th IFAC Symposium on Intelligent Autonomous Vehicles*, 2004.

[14] Patrick Doherty, Gösta Granlund, Krzystof Kuchcinski, Erik Sandewall, Klas Nordberg, Erik Skarman, and Johan Wiklund. The witas unmanned aerial vehicle project. In W.Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence*. IOS Press, Amsterdam, 2000.

[15] Patrick Doherty, Patrik Haslum, Fredrik Heintz, Torsten Merz, Per Nyblom, Tommy Persson, and Björn Wingman. A distributed architecture for autonomous unmanned aerial vehicle experimentation. In *Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems*, 2004.

[16] M. A. Dahleh E. Frazzoli and E. Feron. Robust hybrid control for autonomous vehicles motion planning. Technical Report 2000-4056, AIAA, 1999. `http://gewurtz.lids.mit.edu/papers/FDF99.pdf`, accessed: June 10, 2004.

[17] M. A. Dahleh E. Frazzoli and E. Feron. Real-time motion planning for agile autonomous vehicles. In *AIAA Conf. on Guidance, Navigation and Control*, 2000. `http://gewurtz.mit.edu/papers/FDF00.pdf`, accessed: April 23, 2003.

[18] Etienne Ferré Florent Lamiraux and Erwan Vallée. Kinodynamic motion planning: Connecting exploration trees using trajectory optimization methods, 2004.

[19] E. Frazzoli. *Robust hybrid control for autonomous vehicle motion planning*. PhD thesis, MIT, 2001. `http://gewurtz.lids.mit.edu/papers/F01.pdf`, accessed: June 10, 2004.

[20] H. Goldstein. *Classical Mechanics*. Addison–Wesley, Reading, MA, U.S.A., 2^{nd} edition, 1980.

[21] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. Technical Report TR96-013, Department of Computer Science, University of N. Carolina, Chapel Hill, 1996. URL: `ftp://ftp.cs.unc.edu/pub/users/manocha/PAPERS/COLLISION/obb.ps.gz` accessed 11/05/2001.

[22] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proceedings of IEEE International Conference on Robotics & Automation*, 2003. `http://citeseer.ist.psu.edu/hsu03bridge.html`.

[23] D. Hsu, L. Kavraki, J. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners, 1998. `http://citeseer.ist.psu.edu/hsu98finding.html`.

[24] D. Hsu, L. Kavraki, J.-C. Latombe, and R. Motwani. Capturing the connectivity of high-dimensional geometric spaces by parallelizable random sampling techniques. In P. M. Pardalos and S. Rajasekaran, editors, *Advances in Randomized Parallel Computing*, pages 159–182. Kluwer Academic Publisher, Boston, 1999.

[25] David Hsu, Jean-Claude Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications*, 9(4/5):495–, 1999. `http://citeseer.ist.psu.edu/hsu97path.html`.

[26] L. Jaillet and T. Siméon. A prm-based motion planner for dynamically changing environments. In *Proc. of the IEEE Int. Conf. on Int. Robots and Systems*, 2004. `http://www.laas.fr/~ljaillet/Papers/iros04-dynamic-planner.pdf`, accessed: Friday, August 5, 2005.

[27] M. Kallmann and M. Matarić. Motion planning using dynamic roadmaps. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004. `http://www.ict.usc.edu/`

~kallmann/publications/kallmann_icra_04.pdf, accessed: Friday, August 5, 2005.

[28] L. Kavraki, M. Kolountzakis, and J. Latombe. Analysis of probabilistic roadmaps for path planning, 1996. http://citeseer.ist.psu.edu/article/kavraki98analysis.html.

[29] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996. http://www.cs.rice.edu/CS/Robotics/papers/kavraki1996prm-high-dim-conf.pdf, accessed: November 21, 2003.

[30] Lydia Kavraki and Jean-Claude Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proceedings of the International Conference on Robotics and Automation*, pages 2138–2139, 1994. http://www.cs.rice.edu/CS/Robotics/papers/kavraki1994rand-preproc-fast.pdf, accessed: January 23, 2004.

[31] Lydia E. Kavraki, Jean-Claude Latombe, Rajeev Motwani, and Prabhakar Raghavan. Randomized query processing in robot path planning. In *ACM Symp. on Theory of Computing*, pages 353–362, 1995.
http://citeseer.ist.psu.edu/kavraki96randomized.html.

[32] O Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Rob. Res.*, 5(1):90–98, 1986.
http://ai.stanford.edu/groups/manips/publications.html.

[33] D.g. Kirkpatrick. Efficient computation of continuous skeletons. In *Proceedings of the 20th Symposium on Foundations of Computer Science*, pages 18–27, 1979.

[34] J. J. Kuffner and S. M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *IEEE Int'l Conf. on*

Robotics and Automation, pages 995–1001, 2000.
`http://msl.cs.uiuc.edu/rrt/papers/KufLav00.ps.gz`, accessed: November 12, 2003.

[35] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[36] J.-P. Laumond and T. Siméon. Notes on visibility roadmaps and path planning. Workshop on Algorithmic Foundations of Robotics, 2000. `http://www.laas.fr/~nic/Papers/wafr00.ps.gz`, accessed: January 23, 2004.

[37] S. M. LaValle. *Planning Algorithms.* Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/). To be published in 2006.

[38] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Dept., Iowa State University, Oct. 1998.
`http://msl.cs.uiuc.edu/rrt/papers/Lav98c.ps.gz`, accessed: November 12, 2003.

[39] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *IEEE Int'l Conf. on Robotics and Automation*, pages 473–479, 1999. `http://msl.cs.uiuc.edu/rrt/papers/LavKuf99.ps.gz`, accessed: October 15, 2004.

[40] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, (20(5)):378–400, May 2001.
`http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01b.ps.gz`, accessed: August 5, 2004.

[41] Mark H. Overmars and Petr Švestka. A paradigm for probabilistic path planning. Technical report, Department of Computer Science, Utrecht University, 1996. `http://archive.cs.uu.nl/pub/RUU/CS/techreps/CS-1995/1995-22.pdf`, accessed: January 23, 2004.

[42] M. H. Overmars P. Svestka. Motion planning for car-like robots using a probabilistic learning approach. Technical Report UU-CS-1994-33, Computer Science Department, Utrecht University, 1994. `http://archive.cs.uu.nl/pub/RUU/CS/techreps/CS-1994/1994-33.ps.gz`, accessed: December 10, 2003.

[43] Per Olof Pettersson. Helicopter path planning using probabilistic roadmaps. Master's thesis, Linköpings universitet, January 2003. `http://www.ida.liu.se/~peope/exjobb-y.pdf`, accessed: November 24, 2003.

[44] Per Olof Pettersson and Patrick Doherty. Probabilistic roadmap based path planning for an autonomous unmanned aerial vehicle. ICAPS-04 Workshop on Connecting Planning Theory with Practice, 2004. `http://www.ida.liu.se/~peope/peope-patdo-icaps04.pdf`, accessed: February 16, 2005.

[45] Per Olof Pettersson and Patrick Doherty. Probabilistic roadmap based path planning for an autonomous unmanned helicopter. SAIS-SSLS 2005 Event, 2005. `http://www.ida.liu.se/~peope/SAIS05PetterssonP.pdf`, accessed: February 24, 2005.

[46] Per Olof Pettersson and Patrick Doherty. Probabilistic roadmap based path planning for an autonomous unmanned helicopter. *Journal of Intelligent & Fuzzy Systems: Computational Intelligence in Northern Europe*, 2006. accepted for publication.

[47] J.H. Reif and H. Wang. The complexity of the two dimensional curvature-constrained shortest-path problem. In *Third International Workshop on Algorithmic Foundations of Robotics (WAFR98)*, pages 49–57. A. K. Peters Ltd, Houston, Texas, June 1998. `http://www.cs.duke.edu/~reif/paper/hongyan/curve/curve.pdf`, accessed: September 2, 2004.

[48] S. Sekhavat and J.-P. Laumond. Topological property for collision-free nonholonomic motion planning: the case of sinusoidal

inputs for chained form systems. In *IEEE Trans. Robotics and Automation*, volume 14(5), pages 671–680, October 1998. `http://www.inrialpes.fr/sharp/people/sekhavat/PUBLI/sekhavat:laumond:ieeetra:98.ps`, accessed: December 11, 2003.

[49] S. Sekhavat, J-P. Laumond P. Svestka, and M. H. Overmars. Multi-level path planning for nonholonomic robots using semi-holonomic subsystems. *The international journal of robotics research*, 17:840–857, 1996. `http://archive.cs.uu.nl/pub/RUU/CS/techreps/CS-1996/1996-08.pdf`, accessed: November 24, 2003.

[50] T. Siméon, J. P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps. IEEE Int. Conf. on Int. Robots and Systems, 1999. `http://www.laas.fr/~nic/Papers/iros99.ps.gz`, accessed: January 23, 2004.

[51] T. Siméon, J. P. Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Journal of Advanced Robotics*, 14(6):477–494, 2000. `http://www.laas.fr/~nic/Papers/advrob00.ps.gz`, accessed: January 23, 2004.

[52] Guang Song and Nancy M. Amato. Randomized motion planning for car-like robots with c-prm. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pages 37–42, Nov 2001. `http://parasol-www.cs.tamu.edu/groups/amatogroup/research/carlike/`, accessed: Sunday, July 31, 2005.

Avdelning, Institution
Division, Department

AIICS,
Dept. of Computer and Information Science
581 83 Linköping

Datum
Date

December 29, 2005

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English
☐ ______

Rapporttyp
Report category

☒ Licentiatavhandling
☐ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ ______

ISBN
91–85497–15–0

ISRN
LiU-Tek-Lic–2006:10

Serietitel och serienummer **ISSN**
Title of series, numbering 0280–7971

Linköping Studies in Science and Technology

Thesis No. 1229

URL för elektronisk version
`http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-5270`

Titel
Title

Sampling-based Path Planning for an Autonomous Helicopter

Författare
Author

Per Olof Pettersson

Sammanfattning
Abstract

Many of the applications that have been proposed for future small unmanned aerial vehicles (UAVs) are at low altitude in areas with many obstacles. A vital component for successful navigation in such environments is a path planner that can find collision free paths for the UAV.

Two popular path planning algorithms are the probabilistic roadmap algorithm (PRM) and the rapidly-exploring random tree algorithm (RRT). Adaptations of these algorithms to an unmanned autonomous helicopter are presented in this thesis, together with a number of extensions for handling constraints at different stages of the planning process.

The result of this work is twofold:

First, the described planners and extensions have been implemented and integrated into the software architecture of a UAV. A number of flight tests with these algorithms have been performed on a physical helicopter and the results from some of them are presented in this thesis.

Second, an empirical study has been conducted, comparing the performance of the different algorithms and extensions in this planning domain. It is shown that with the environment known in advance, the PRM algorithm generally performs better than the RRT algorithm due to its precompiled roadmaps, but that the latter is also usable as long as the environment is not too complex. The study also shows that simple geometric constraints can be added in the runtime phase of the PRM algorithm, without a big impact on performance. It is also shown that postponing the motion constraints to the runtime phase can improve the performance of the planner in some cases.

Nyckelord
Keywords

Path Planning, Motion Planning, Helicopters, Probabilistic Roadmaps, Rapidly-exploring Random Trees

Department of Computer and Information Science
Linköpings universitet

Linköping Studies in Science and Technology
Faculty of Arts and Sciences - Licentiate Theses

No 17 **Vojin Plavsic:** Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)
No 28 **Arne Jönsson, Mikael Patel:** An Interactive Flowcharting Technique for Communicating and Realizing Algorithms, 1984.
No 29 **Johnny Eckerland:** Retargeting of an Incremental Code Generator, 1984.
No 48 **Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching, 1985.
No 52 **Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems, 1985.
No 60 **Johan Fagerström:** Simulation and Evaluation of Architecture based on Asynchronous Processes, 1985.
No 71 **Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System, 1987.
No 72 **Tony Larsson:** On the Specification and Verification of VLSI Systems, 1986.
No 73 **Ola Strömfors:** A Structure Editor for Documents and Programs, 1986.
No 74 **Christos Levcopoulos:** New Results about the Approximation Behavior of the Greedy Triangulation, 1986.
No 104 **Shamsul I. Chowdhury: S**tatistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology, 1987.
No 108 **Rober Bilos:** Incremental Scanning and Token-Based Editing, 1987.
No 111 **Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments, 1987.
No 113 **Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge, 1987.
No 118 **Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs, 1987.
No 126 **Dan Strömberg:** Transfer and Distribution of Application Programs, 1987.
No 127 **Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems, 1987.
No 139 **Christer Bäckström:** Reasoning about Interdependent Actions, 1988.
No 140 **Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing, 1988.
No 146 **Johan Hultman:** A Software System for Defining and Controlling Actions in a Mechanical System, 1988.
No 150 **Tim Hansen:** Diagnosing Faults using Knowledge about Malfunctioning Behavior, 1988.
No 165 **Jonas Löwgren:** Supporting Design and Management of Expert System User Interfaces, 1989.
No 166 **Ola Petersson:** On Adaptive Sorting in Sequential and Parallel Models, 1989.
No 174 **Yngve Larsson:** Dynamic Configuration in a Distributed Environment, 1989.
No 177 **Peter Åberg:** Design of a Multiple View Presentation and Interaction Manager, 1989.
No 181 **Henrik Eriksson:** A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.
No 184 **Ivan Rankin:** The Deep Generation of Text in Expert Critiquing Systems, 1989.
No 187 **Simin Nadjm-Tehrani:** Contributions to the Declarative Approach to Debugging Prolog Programs, 1989.
No 189 **Magnus Merkel:** Temporal Information in Natural Language, 1989.
No 196 **Ulf Nilsson:** A Systematic Approach to Abstract Interpretation of Logic Programs, 1989.
No 197 **Staffan Bonnier:** Horn Clause Logic with External Procedures: Towards a Theoretical Framework, 1989.
No 203 **Christer Hansson:** A Prototype System for Logical Reasoning about Time and Action, 1990.
No 212 **Björn Fjellborg:** An Approach to Extraction of Pipeline Structures for VLSI High-Level Synthesis, 1990.
No 230 **Patrick Doherty:** A Three-Valued Approach to Non-Monotonic Reasoning, 1990.
No 237 **Tomas Sokolnicki:** Coaching Partial Plans: An Approach to Knowledge-Based Tutoring, 1990.
No 250 **Lars Strömberg:** Postmortem Debugging of Distributed Systems, 1990.
No 253 **Torbjörn Näslund:** SLDFA-Resolution - Computing Answers for Negative Queries, 1990.
No 260 **Peter D. Holmes:** Using Connectivity Graphs to Support Map-Related Reasoning, 1991.
No 283 **Olof Johansson:** Improving Implementation of Graphical User Interfaces for Object-Oriented Knowledge-Bases, 1991.
No 298 **Rolf G Larsson:** Aktivitetsbaserad kalkylering i ett nytt ekonomisystem, 1991.
No 318 **Lena Srömbäck:** Studies in Extended Unification-Based Formalism for Linguistic Description: An Algorithm for Feature Structures with Disjunction and a Proposal for Flexible Systems, 1992.
No 319 **Mikael Pettersson:** DML-A Language and System for the Generation of Efficient Compilers from Denotational Specification, 1992.
No 326 **Andreas Kågedal:** Logic Programming with External Procedures: an Implementation, 1992.
No 328 **Patrick Lambrix:** Aspects of Version Management of Composite Objects, 1992.
No 333 **Xinli Gu:** Testability Analysis and Improvement in High-Level Synthesis Systems, 1992.
No 335 **Torbjörn Näslund:** On the Role of Evaluations in Iterative Development of Managerial Support Sytems, 1992.
No 348 **Ulf Cederling:** Industrial Software Development - a Case Study, 1992.
No 352 **Magnus Morin:** Predictable Cyclic Computations in Autonomous Systems: A Computational Model and Implementation, 1992.
No 371 **Mehran Noghabai:** Evaluation of Strategic Investments in Information Technology, 1993.
No 378 **Mats Larsson:** A Transformational Approach to Formal Digital System Design, 1993.
No 380 **Johan Ringström:** Compiler Generation for Parallel Languages from Denotational Specifications, 1993.
No 381 **Michael Jansson:** Propagation of Change in an Intelligent Information System, 1993.
No 383 **Jonni Harrius:** An Architecture and a Knowledge Representation Model for Expert Critiquing Systems, 1993.
No 386 **Per Österling:** Symbolic Modelling of the Dynamic Environments of Autonomous Agents, 1993.
No 398 **Johan Boye:** Dependency-based Groudness Analysis of Functional Logic Programs, 1993.

No 402	**Lars Degerstedt:** Tabulated Resolution for Well Founded Semantics, 1993.
No 406	**Anna Moberg:** Satellitkontor - en studie av kommunikationsmönster vid arbete på distans, 1993.
No 414	**Peter Carlsson:** Separation av företagsledning och finansiering - fallstudier av företagsledarutköp ur ett agent-teoretiskt perspektiv, 1994.
No 417	**Camilla Sjöström:** Revision och lagreglering - ett historiskt perspektiv, 1994.
No 436	**Cecilia Sjöberg:** Voices in Design: Argumentation in Participatory Development, 1994.
No 437	**Lars Viklund:** Contributions to a High-level Programming Environment for a Scientific Computing, 1994.
No 440	**Peter Loborg:** Error Recovery Support in Manufacturing Control Systems, 1994.
FHS 3/94	**Owen Eriksson:** Informationssystem med verksamhetskvalitet - utvärdering baserat på ett verksamhetsinrik-tat och samskapande perspektiv, 1994.
FHS 4/94	**Karin Pettersson:** Informationssystemstrukturering, ansvarsfördelning och användarinflytande - En kompa-rativ studie med utgångspunkt i två informationssystemstrategier, 1994.
No 441	**Lars Poignant:** Informationsteknologi och företagsetablering - Effekter på produktivitet och region, 1994.
No 446	**Gustav Fahl:** Object Views of Relational Data in Multidatabase Systems, 1994.
No 450	**Henrik Nilsson:** A Declarative Approach to Debugging for Lazy Functional Languages, 1994.
No 451	**Jonas Lind:** Creditor - Firm Relations: an Interdisciplinary Analysis, 1994.
No 452	**Martin Sköld:** Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques, 1994.
No 455	**Pär Carlshamre:** A Collaborative Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, 1994.
FHS 5/94	**Stefan Cronholm:** Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende ar-betssätt och arbetsformer, 1994.
No 462	**Mikael Lindvall:** A Study of Traceability in Object-Oriented Systems Development, 1994.
No 463	**Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av Sandviks förvärv av Bahco Verktyg, 1994.
No 464	**Hans Olsén:** Collage Induction: Proving Properties of Logic Programs by Program Synthesis, 1994.
No 469	**Lars Karlsson:** Specification and Synthesis of Plans Using the Features and Fluents Framework, 1995.
No 473	**Ulf Söderman:** On Conceptual Modelling of Mode Switching Systems, 1995.
No 475	**Choong-ho Yi:** Reasoning about Concurrent Actions in the Trajectory Semantics, 1995.
No 476	**Bo Lagerström:** Successiv resultatavräkning av pågående arbeten. - Fallstudier i tre byggföretag, 1995.
No 478	**Peter Jonsson:** Complexity of State-Variable Planning under Structural Restrictions, 1995.
FHS 7/95	**Anders Avdic:** Arbetsintegrerad systemutveckling med kalkylkprogram, 1995.
No 482	**Eva L Ragnemalm:** Towards Student Modelling through Collaborative Dialogue with a Learning Compani-on, 1995.
No 488	**Eva Toller:** Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
No 489	**Erik Stoy:** A Petri Net Based Unified Representation for Hardware/Software Co-Design, 1995.
No 497	**Johan Herber:** Environment Support for Building Structured Mathematical Models, 1995.
No 498	**Stefan Svenberg:** Structure-Driven Derivation of Inter-Lingual Functor-Argument Trees for Multi-Lingual Generation, 1995.
No 503	**Hee-Cheol Kim:** Prediction and Postdiction under Uncertainty, 1995.
FHS 8/95	**Dan Fristedt:** Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
FHS 9/95	**Malin Bergvall:** Systemförvaltning i praktiken - en kvalitativ studie avseende centrala begrepp, aktiviteter och ansvarsroller, 1995.
No 513	**Joachim Karlsson:** Towards a Strategy for Software Requirements Selection, 1995.
No 517	**Jakob Axelsson:** Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems, 1995.
No 518	**Göran Forslund:** Toward Cooperative Advice-Giving Systems: The Expert Systems Experience, 1995.
No 522	**Jörgen Andersson:** Bilder av småföretagares ekonomistyrning, 1995.
No 538	**Staffan Flodin:** Efficient Management of Object-Oriented Queries with Late Binding, 1996.
No 545	**Vadim Engelson:** An Approach to Automatic Construction of Graphical User Interfaces for Applications in Scientific Computing, 1996.
No 546	**Magnus Werner :** Multidatabase Integration using Polymorphic Queries and Views, 1996.
FiF-a 1/96	**Mikael Lind:** Affärsprocessinriktad förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.
No 549	**Jonas Hallberg:** High-Level Synthesis under Local Timing Constraints, 1996.
No 550	**Kristina Larsen:** Förutsättningar och begränsningar för arbete på distans - erfarenheter från fyra svenska före-tag. 1996.
No 557	**Mikael Johansson:** Quality Functions for Requirements Engineering Methods, 1996.
No 558	**Patrik Nordling:** The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
No 561	**Anders Ekman:** Exploration of Polygonal Environments, 1996.
No 563	**Niclas Andersson:** Compilation of Mathematical Models to Parallel Code, 1996.
No 567	**Johan Jenvald:** Simulation and Data Collection in Battle Training, 1996.
No 575	**Niclas Ohlsson:** Software Quality Engineering by Early Identification of Fault-Prone Modules, 1996.
No 576	**Mikael Ericsson:** Commenting Systems as Design Support—A Wizard-of-Oz Study, 1996.
No 587	**Jörgen Lindström:** Chefers användning av kommunikationsteknik, 1996.
No 589	**Esa Falkenroth:** Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.
No 591	**Niclas Wahllöf:** A Default Extension to Description Logics and its Applications, 1996.
No 595	**Annika Larsson:** Ekonomisk Styrning och Organisatorisk Passion - ett interaktivt perspektiv, 1997.
No 597	**Ling Lin:** A Value-based Indexing Technique for Time Sequences, 1997.

No 598	**Rego Granlund:** C^3Fire - A Microworld Supporting Emergency Management Training, 1997.
No 599	**Peter Ingels:** A Robust Text Processing Technique Applied to Lexical Error Recovery, 1997.
No 607	**Per-Arne Persson:** Toward a Grounded Theory for Support of Command and Control in Military Coalitions, 1997.
No 609	**Jonas S Karlsson:** A Scalable Data Structure for a Parallel Data Server, 1997.
FiF-a 4	**Carita Åbom:** Videomötesteknik i olika affärssituationer - möjligheter och hinder, 1997.
FiF-a 6	**Tommy Wedlund**: Att skapa en företagsanpassad systemutvecklingsmodell - genom rekonstruktion, värdering och vidareutveckling i T50-bolag inom ABB, 1997.
No 615	**Silvia Coradeschi**: A Decision-Mechanism for Reactive and Coordinated Agents, 1997.
No 623	**Jan Ollinen:** Det flexibla kontorets utveckling på Digital - Ett stöd för multiflex? 1997.
No 626	**David Byers:** Towards Estimating Software Testability Using Static Analysis, 1997.
No 627	**Fredrik Eklund:** Declarative Error Diagnosis of GAPLog Programs, 1997.
No 629	**Gunilla Ivefors:** Krigsspel coh Informationsteknik inför en oförutsägbar framtid, 1997**.**
No 631	**Jens-Olof Lindh:** Analysing Traffic Safety from a Case-Based Reasoning Perspective, 1997
No 639	**Jukka Mäki-Turja:**. Smalltalk - a suitable Real-Time Language, 1997.
No 640	**Juha Takkinen**: CAFE: Towards a Conceptual Model for Information Management in Electronic Mail, 1997.
No 643	**Man Lin**: Formal Analysis of Reactive Rule-based Programs, 1997.
No 653	**Mats Gustafsson**: Bringing Role-Based Access Control to Distributed Systems, 1997.
FiF-a 13	**Boris Karlsson:** Metodanalys för förståelse och utveckling av systemutvecklingsverksamhet. Analys och värdering av systemutvecklingsmodeller och dess användning, 1997.
No 674	**Marcus Bjäreland:** Two Aspects of Automating Logics of Action and Change - Regression and Tractability, 1998.
No 676	**Jan Håkegård**: Hiera rchical Test Architecture and Board-Level Test Controller Synthesis, 1998.
No 668	**Per-Ove Zetterlund**: Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998.
No 675	**Jimmy Tjäder**: Projektledaren & planen - en studie av projektledning i tre installations- och systemutvecklingsprojekt, 1998.
FiF-a 14	**Ulf Melin**: Informationssystem vid ökad affärs- och processorientering - egenskaper, strategier och utveckling, 1998.
No 695	**Tim Heyer**: COMPASS: Introduction of Formal Methods in Code Development and Inspection, 1998.
No 700	**Patrik Hägglund:** Programming Languages for Computer Algebra, 1998.
FiF-a 16	**Marie-Therese Christiansson:** Inter-organistorisk verksamhetsutveckling - metoder som stöd vid utveckling av partnerskap och informationssystem, 1998.
No 712	**Christina Wennestam:** Information om immateriella resurser. Investeringar i forskning och utveckling samt i personal inom skogsindustrin, 1998.
No 719	**Joakim Gustafsson:** Extending Temporal Action Logic for Ramification and Concurrency, 1998.
No 723	**Henrik André-Jönsson:** Indexing time-series data using text indexing methods, 1999.
No 725	**Erik Larsson:** High-Level Testability Analysis and Enhancement Techniques, 1998.
No 730	**Carl-Johan Westin:** Informationsförsörjning: en fråga om ansvar - aktiviteter och uppdrag i fem stora svenska organisationers operativa informationsförsörjning, 1998.
No 731	**Åse Jansson:** Miljöhänsyn - en del i företags styrning, 1998.
No 733	**Thomas Padron-McCarthy:** Performance-Polymorphic Declarative Queries, 1998.
No 734	**Anders Bäckström:** Värdeskapande kreditgivning - Kreditriskhantering ur ett agentteoretiskt perspektiv, 1998.
FiF-a 21	**Ulf Seigerroth:** Integration av förändringsmetoder - en modell för välgrundad metodintegration, 1999.
FiF-a 22	**Fredrik Öberg:** Object-Oriented Frameworks - A New Strategy for Case Tool Development, 1998.
No 737	**Jonas Mellin:** Predictable Event Monitoring, 1998.
No 738	**Joakim Eriksson:** Specifying and Managing Rules in an Active Real-Time Database System, 1998.
FiF-a 25	**Bengt E W Andersson:** Samverkande informationssystem mellan aktörer i offentliga åtaganden - En teori om aktörsarenor i samverkan om utbyte av information, 1998.
No 742	**Pawel Pietrzak:** Static Incorrectness Diagnosis of CLP (FD), 1999.
No 748	**Tobias Ritzau:** Real-Time Reference Counting in RT-Java, 1999.
No 751	**Anders Ferntoft:** Elektronisk affärskommunikation - kontaktkostnader och kontaktprocesser mellan kunder och leverantörer på producentmarknader,1999.
No 752	**Jo Skåmedal:** Arbete på distans och arbetsformens påverkan på resor och resmönster, 1999.
No 753	**Johan Alvehus:** Mötets metaforer. En studie av berättelser om möten, 1999.
No 754	**Magnus Lindahl:** Bankens villkor i låneavtal vid kreditgivning till högt belånade företagsförvärv: En studie ur ett agentteoretiskt perspektiv, 2000.
No 766	**Martin V. Howard:** Designing dynamic visualizations of temporal data, 1999.
No 769	**Jesper Andersson:** Towards Reactive Software Architectures, 1999.
No 775	**Anders Henriksson:** Unique kernel diagnosis, 1999.
FiF-a 30	**Pär J. Ågerfalk:** Pragmatization of Information Systems - A Theoretical and Methodological Outline, 1999.
No 787	**Charlotte Björkegren:** Learning for the next project - Bearers and barriers in knowledge transfer within an organisation, 1999.
No 788	**Håkan Nilsson:** Informationsteknik som drivkraft i granskningsprocessen - En studie av fyra revisionsbyråer, 2000.
No 790	**Erik Berglund:** Use-Oriented Documentation in Software Development, 1999.
No 791	**Klas Gäre:** Verksamhetsförändringar i samband med IS-införande, 1999.
No 800	**Anders Subotic:** Software Quality Inspection, 1999.
No 807	**Svein Bergum**: Managerial communication in telework, 2000.

No 809	**Flavius Gruian:** Energy-Aware Design of Digital Systems, 2000.
FiF-a 32	**Karin Hedström:** Kunskapsanvändning och kunskapsutveckling hos verksamhetskonsulter - Erfarenheter från ett FOU-samarbete, 2000.
No 808	**Linda Askenäs:** Affärssystemet - En studie om teknikens aktiva och passiva roll i en organisation, 2000.
No 820	**Jean Paul Meynard:** Control of industrial robots through high-level task programming, 2000.
No 823	**Lars Hult:** Publika Gränsytor - ett designexempel, 2000.
No 832	**Paul Pop:** Scheduling and Communication Synthesis for Distributed Real-Time Systems, 2000.
FiF-a 34	**Göran Hultgren:** Nätverksinriktad Förändringsanalys - perspektiv och metoder som stöd för förståelse och utveckling av affärsrelationer och informationssystem, 2000.
No 842	**Magnus Kald:** The role of management control systems in strategic business units, 2000.
No 844	**Mikael Cäker:** Vad kostar kunden? Modeller för intern redovisning, 2000.
FiF-a 37	**Ewa Braf**: Organisationers kunskapsverksamheter - en kritisk studie av ”knowledge management”, 2000.
FiF-a 40	**Henrik Lindberg:** Webbaserade affärsprocesser - Möjligheter och begränsningar, 2000.
FiF-a 41	**Benneth Christiansson:** Att komponentbasera informationssystem - Vad säger teori och praktik?, 2000.
No. 854	**Ola Pettersson:** Deliberation in a Mobile Robot, 2000.
No 863	**Dan Lawesson**: Towards Behavioral Model Fault Isolation for Object Oriented Control Systems, 2000.
No 881	**Johan Moe:** Execution Tracing of Large Distributed Systems, 2001.
No 882	**Yuxiao Zhao:** XML-based Frameworks for Internet Commerce and an Implementation of B2B e-procurement, 2001.
No 890	**Annika Flycht-Eriksson:** Domain Knowledge Management inInformation-providing Dialogue systems, 2001.
Fif-a 47	**Per-Arne Segerkvis**t: Webbaserade imaginära organisationers samverkansformer, 2001.
No 894	**Stefan Svarén:** Styrning av investeringar i divisionaliserade företag - Ett koncernperspektiv, 2001.
No 906	**Lin Han**: Secure and Scalable E-Service Software Delivery, 2001.
No 917	**Emma Hansson:** Optionsprogram för anställda - en studie av svenska börsföretag, 2001.
No 916	**Susanne Odar:** IT som stöd för strategiska beslut, en studie av datorimplementerade modeller av verksamhet som stöd för beslut om anskaffning av JAS 1982, 2002.
Fif-a-49	**Stefan Holgersson:** IT-system och filtrering av verksamhetskunskap - kvalitetsproblem vid analyser och beslutsfattande som bygger på uppgifter hämtade från polisens IT-system, 2001.
Fif-a-51	**Per Oscarsson:**Informationssäkerhet i verksamheter - begrepp och modeller som stöd för förståelse av informationssäkerhet och dess hantering, 2001.
No 919	**Luis Alejandro Cortes:** A Petri Net Based Modeling and Verification Technique for Real-Time Embedded Systems, 2001.
No 915	**Niklas Sandell:** Redovisning i skuggan av en bankkris - Värdering av fastigheter. 2001.
No 931	**Fredrik Elg:** Ett dynamiskt perspektiv på individuella skillnader av heuristisk kompetens, intelligens, mentala modeller, mål och konfidens i kontroll av mikrovärlden Moro, 2002.
No 933	**Peter Aronsson:** Automatic Parallelization of Simulation Code from Equation Based Simulation Languages, 2002.
No 938	**Bourhane Kadmiry**: Fuzzy Control of Unmanned Helicopter, 2002.
No 942	**Patrik Haslum**: Prediction as a Knowledge Representation Problem: A Case Study in Model Design, 2002.
No 956	**Robert Sevenius:** On the instruments of governance - A law & economics study of capital instruments in limited liability companies, 2002.
FiF-a 58	**Johan Petersson:** Lokala elektroniska marknadsplatser - informationssystem för platsbundna affärer, 2002.
No 964	**Peter Bunus:** Debugging and Structural Analysis of Declarative Equation-Based Languages, 2002.
No 973	**Gert Jervan:** High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems, 2002.
No 958	**Fredrika Berglund:** Management Control and Strategy - a Case Study of Pharmaceutical Drug Development, 2002.
Fif-a 61	**Fredrik Karlsson:** Meta-Method for Method Configuration - A Rational Unified Process Case, 2002.
No 985	**Sorin Manolache:** Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times, 2002.
No 982	**Diana Szentiványi:** Performance and Availability Trade-offs in Fault-Tolerant Middleware, 2002.
No 989	**Iakov Nakhimovski:** Modeling and Simulation of Contacting Flexible Bodies in Multibody Systems, 2002.
No 990	**Levon Saldamli:** PDEModelica - Towards a High-Level Language for Modeling with Partial Differential Equations, 2002.
No 991	**Almut Herzog:** Secure Execution Environment for Java Electronic Services, 2002.
No 999	**Jon Edvardsson:** Contributions to Program- and Specification-based Test Data Generation, 2002
No 1000	**Anders Arpteg:** Adaptive Semi-structured Information Extraction, 2002.
No 1001	**Andrzej Bednarski:** A Dynamic Programming Approach to Optimal Retargetable Code Generation for Irregular Architectures, 2002.
No 988	**Mattias Arvola:** Good to use! : Use quality of multi-user applications in the home, 2003.
FiF-a 62	**Lennart Ljung:** Utveckling av en projektivitetsmodell - om organisationers förmåga att tillämpa projektarbetsformen, 2003.
No 1003	**Pernilla Qvarfordt:** User experience of spoken feedback in multimodal interaction, 2003.
No 1005	**Alexander Siemers:** Visualization of Dynamic Multibody Simulation With Special Reference to Contacts, 2003.
No 1008	**Jens Gustavsson:** Towards Unanticipated Runtime Software Evolution, 2003.
No 1010	**Calin Curescu:** Adaptive QoS-aware Resource Allocation for Wireless Networks, 2003.
No 1015	**Anna Andersson:** Management Information Systems in Process-oriented Healthcare Organisations, 2003.
No 1018	**Björn Johansson:** Feedforward Control in Dynamic Situations, 2003.
No 1022	**Traian Pop:** Scheduling and Optimisation of Heterogeneous Time/Event-Triggered Distributed Embedded Systems, 2003.
FiF-a 65	**Britt-Marie Johansson:** Kundkommunikation på distans - en studie om kommunikationsmediets betydelse i affärstransaktioner, 2003.
No 1024	**Aleksandra Tesanovic:** Towards Aspectual Component-Based Real-Time System Development, 2003.

No 1034	**Arja Vainio-Larsson**: Designing for Use in a Future Context - Five Case Studies in Retrospect, 2003.
No 1033	**Peter Nilsson:** Svenska bankers redovisningsval vid reservering för befarade kreditförluster - En studie vid införandet av nya redovisningsregler, 2003.
Fif-a 69	**Fredrik Ericsson:** Information Technology for Learning and Acquiring of Work Knowledge, 2003.
No 1049	**Marcus Comstedt:** Towards Fine-Grained Binary Composition through Link Time Weaving, 2003.
No 1052	**Åsa Hedenskog:** Increasing the Automation of Radio Network Control, 2003.
No 1054	**Claudiu Duma:** Security and Efficiency Tradeoffs in Multicast Group Key Management, 2003.
Fif-a 71	**Emma Eliasson:** Effektanalys av IT-systems handlingsutrymme, 2003.
No 1055	**Carl Cederberg:** Experiments in Indirect Fault Injection with Open Source and Industrial Software, 2003.
No 1058	**Daniel Karlsson:** Towards Formal Verification in a Component-based Reuse Methodology, 2003.
FiF-a 73	**Anders Hjalmarsson:** Att etablera och vidmakthålla förbättringsverksamhet - behovet av koordination och interaktion vid förändring av systemutvecklingsverksamheter, 2004.
No 1079	**Pontus Johansson:** Design and Development of Recommender Dialogue Systems, 2004.
No 1084	**Charlotte Stoltz:** Calling for Call Centres - A Study of Call Centre Locations in a Swedish Rural Region, 2004.
FiF-a 74	**Björn Johansson:** Deciding on Using Application Service Provision in SMEs, 2004.
No 1094	**Genevieve Gorrell:** Language Modelling and Error Handling in Spoken Dialogue Systems, 2004.
No 1095	**Ulf Johansson:** Rule Extraction - the Key to Accurate and Comprehensible Data Mining Models, 2004.
No 1099	**Sonia Sangari:** Computational Models of Some Communicative Head Movements, 2004.
No 1110	**Hans Nässla:** Intra-Family Information Flow and Prospects for Communication Systems, 2004.
No 1116	**Henrik Sällberg:** On the value of customer loyalty programs - A study of point programs and switching costs, 2004.
FiF-a 77	**Ulf Larsson:** Designarbete i dialog - karaktärisering av interaktionen mellan användare och utvecklare i en systemutvecklingsprocess, 2004.
No 1126	**Andreas Borg:** Contribution to Management and Validation of Non-Functional Requirements, 2004.
No 1127	**Per-Ola Kristensson:** Large Vocabulary Shorthand Writing on Stylus Keyboard, 2004.
No 1132	**Pär-Anders Albinsson:** Interacting with Command and Control Systems: Tools for Operators and Designers, 2004.
No 1130	**Ioan Chisalita:** Safety-Oriented Communication in Mobile Networks for Vehicles, 2004.
No 1138	**Thomas Gustafsson:** Maintaining Data Consistency im Embedded Databases for Vehicular Systems, 2004.
No 1149	**Vaida Jakonienė:** A Study in Integrating Multiple Biological Data Sources, 2005.
No 1156	**Abdil Rashid Mohamed**: High-Level Techniques for Built-In Self-Test Resources Optimization, 2005.
No 1162	**Adrian Pop:** Contributions to Meta-Modeling Tools and Methods, 2005.
No 1165	**Fidel Vascós Palacios:** On the information exchange between physicians and social insurance officers in the sick leave process: an Activity Theoretical perspective, 2005.
FiF-a 84	**Jenny Lagsten:** Verksamhetsutvecklande utvärdering i informationssystemprojekt, 2005.
No 1166	**Emma Larsdotter Nilsson:** Modeling, Simulation, and Visualization of Metabolic Pathways Using Modelica, 2005.
No 1167	**Christina Keller:** Virtual Learning Environments in higher education. A study of students' acceptance of educational technology, 2005.
No 1168	**Cécile Åberg:** Integration of organizational workflows and the Semantic Web, 2005.
FiF-a 85	**Anders Forsman:** Standardisering som grund för informationssamverkan och IT-tjänster - En fallstudie baserad på trafikinformationstjänsten RDS-TMC, 2005.
No 1171	**Yu-Hsing Huang:** A systemic traffic accident model, 2005.
FiF-a 86	**Jan Olausson:** Att modellera uppdrag - grunder för förståelse av processinriktade informationssystem i transaktionsintensiva verksamheter, 2005.
No 1172	**Petter Ahlström**: Affärsstrategier för seniorbostadsmarknaden, 2005.
No 1183	**Mathias Cöster**: Beyond IT and Productivity - How Digitization Transformed the Graphic Industry, 2005.
No 1184	**Åsa Horzella**: Beyond IT and Productivity - Effects of Digitized Information Flows in Grocery Distribution, 2005.
No 1185	**Maria Kollberg**: Beyond IT and Productivity - Effects of Digitized Information Flows in the Logging Industry, 2005.
No 1190	**David Dinka**: Role and Identity - Experience of technology in professional settings, 2005.
No 1191	**Andreas Hansson**: Increasing the Storage Capacity of Recursive Auto-associative Memory by Segmenting Data, 2005.
No 1192	**Nicklas Bergfeldt:** Towards Detached Communication for Robot Cooperation, 2005.
No 1194	**Dennis Maciuszek:** Towards Dependable Virtual Companions for Later Life, 2005.
No 1204	**Beatrice Alenljung:** Decision-making in the Requirements Engineering Process: A Human-centered Approach, 2005
No 1206	**Anders Larsson:** System-on-Chip Test Scheduling and Test Infrastructure Design, 2005.
No 1207	**John Wilander:** Policy and Implementation Assurance for Software Security, 2005.
No 1209	**Andreas Käll:** Översättningar av en managementmodell - En studie av införandet av Balanced Scorecard i ett landsting, 2005.
No 1225	**He Tan:** Aligning and Merging Biomedical Ontologies, 2006.
No 1228	**Artur Wilk:** Descriptive Types for XML Query Language Xcerpt, 2006.
No 1229	**Per Olof Pettersson:** Sampling-based Path Planning for an Autonomous Helicopter, 2006.