Helicopter Path Planning using Probabilistic Roadmaps

Master's thesis
Artificial Intelligence &
Integrated Computer Systems Division

by
Per Olof Pettersson

Reg nr: LiTH-IDA-EX-02-56

January 28, 2003

Helicopter Path Planning using Probabilistic Roadmaps

Master's thesis

performed in **Artificial Intelligence & Integrated Computer Systems Division**, **Dept. of Computer Science** at **Linköpings universitet**

by **Per Olof Pettersson**

Reg nr: LiTH-IDA-EX-02-56

Supervisor: **Professor Patrick Doherty**
Department of Computer Science
at Linköpings universitet

Examiner: **Professor Patrick Doherty**
Department of Computer Science
at Linköpings universitet

Linköping, January 28, 2003

Abstract

Probabilistic roadmap algorithms work in two stages. In the first, by randomly choosing collision free configurations of a robot and then attempting to connect these, a graph is created which represents the free space of the robot's environment. This graph can then be used to answer path planning queries during runtime.

In this report I describe an adaptation of this algorithm to the domain of an autonomously flying helicopter. I also present two extensions to the algorithm: visibility based probabilistic roadmaps and a method of initially relaxing the non-holonomic constraints of the robot.

The probabilistic roadmap algorithm requires a local path planner that is able to plan paths that take the specific robot, in this case a helicopter, from one state to a nearby one. In this report I present two variants of such a local planner that generates a flight path described as a parameterized cubic curve.

A third component necessary for path planning is a method for checking if a certain path is collision free. For this purpose a divide and conquer algorithm was used that bounds the whole model in one box and then recursively splits the box into smaller boxes. A path can then be checked for collision by solving for the intersections of the curve and the planes bounding these boxes, recursing down the tree-structure.

I show that this type of algorithm can successfully give answers to path planning queries in this domain, especially with the initial relaxation of non-holonomic constraints, which makes it possible to find paths even through very narrow passages.

Keywords: Path planning, Probabilistic Roadmaps, Collision checking, OBBTree, Helicopters

Acknowledgments

The work presented in this report 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 the University of Linköping

First I would like to thank my supervisor Patrick Doherty who has given me the opportunity to work with such an exciting project as the WITAS-project, and his confidence in my ability to provide a working solution for the path planner component of the WITAS helicopter.

I would also like to thank the following persons who have given of their time to explain different aspects of the WITAS-project and helped me out with problems that have arisen during the course of the work: Erik Skarman for explaining the different aspects of helicopter mechanics and control issues, and Tommy Persson, Björn Wingman and Patrik Haslum for helping me with the integration of the path planner with the rest of the WITAS software.

Finally I would like to give my thanks to the Knut and Alice Wallenberg foundation for funding the WITAS research project within which this thesis work has been conducted.

Contents

Chapter 1

Introduction

During this thesis work I have investigated the problem of finding a path that takes an autonomous helicopter from one place to another, while avoiding obstacles in the environment along the way. The method that I have investigated is based on the probabilistic roadmap algorithm, a modern path planning method based on a randomized sampling of the configuration space which is used to construct a graph representing the possible paths in the world. With this graph the path planning problem can be solved using conventional methods of graph-traversal during mission time.

This research has been conducted within the WITAS project. In this chapter I will give a brief description of this project together with an introduction to the topic of path-planning. In the following chapters I will take up the three main components of the chosen path-planning approach: local path planning, collision checking and the probabilistic roadmap algorithm. Conclusions are provided in chapter 5 with a discussion of some topics I consider worthy of further research.

1.1 Problem formulation

The purpose of this thesis has been to investigate the use of probabilistic roadmaps as one method for path planning for an autonomous helicopter used in the WITAS unmanned aerial vehicle project. The goal has also been to develop a functional path planner that can be used in the project for practical experiments with the helicopter and its simulated counterpart. This implies that it has to be able to rapidly and efficiently, generate a feasible flight path between two points that does not deviate too much from the optimal path. Furthermore, the path planner has to be integrated with the WITAS UAV software platform and use the control system of the actual helicopter.

For this basic study a number of simplifying assumptions have been made. The world is assumed to be static without any moving obstacles such as other vehicles that would have to be avoided. It is also assumed that the path planning algorithm has a complete description of the world in advance in the form of a 3D model.

1.2 The WITAS-project

The WITAS (vee-tas) UAV project [3, 4] is an ambitious research project focused on the development of information technology for unmanned aerial vehicles and its combination with low-level control and hardware platforms. The long term goal of the project is the development of a fully autonomous unmanned helicopter which can be used in applications involving photogrammetry, surveillance and 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, GIS, simulation tools, multimodal ground operator interfaces to the UAV and much more. Figure 1.1 is a picture of the Yamaha RMAX helicopter platform used in the project.

The main body of research is 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 is done in cooperation with the Vision Laboratory at the Department of Electrical Engineering, also at Linköping University. In addition, there is cooperation with a number of other different research groups both within Sweden and internationally.

Figure 1.1: The Yamaha RMAX helicopter

1.3 Path planning

The path planning problem refers to the problem of finding a series of movements, achievable by a robot, that take it from one state to another.

1.3.1 Basic Concepts

The movements of a robot are commonly analyzed in terms of its degrees of freedom which are the variables required to fully specify its external mechanical state. For a rigid robot, this would mean its orientation and position in space, while the degrees of freedom for articulated robots also include the position of each joint.

The state-variables can be viewed as spanning a *configuration space* that will be denoted by C, which is the space of all possible configurations of the robot. Some combinations of values for the robot state are not attainable by the robot due to the presence of obstacles that hinder the robot from reaching certain positions, or from constraints on how different links of a non-rigid robot can be moved in relation to each other. The subset of the configuration space that it is possible for the robot to visit is called *free space*, and will be denoted C_f.

The problem of path planning can now be defined as the construction of a continuous curve in C_f that connects an initial configuration and a goal configuration. If the robot is always free to move in any direction in configuration space (as long as it stays in C_f) , it is said to be holonomic. An example of this is a robotic arm where each joint can be bent individually and simultaneously. For the class of holonomic robots any continuous curve in C_f that connects the two configurations is a solution.

However, there also exist a large class of problems that do not have this property, i.e. the robot can not change state in every direction in the configuration space. These robots are said to be non-holonomic. An example of a non-holonomic robot is a car-like robot that can only go backwards and forwards and turn with a certain minimum radius. It is easy to see that this kind of car-like robot is non-holonomic by noticing that the car is unable to move sideways. The constraints on motion for non-holonomic robots make the path-planning problem much harder to solve since, in many cases, not all continuous curves through C_f are solutions.

1.3.2 Probabilistic Roadmaps

The problem of finding an optimal path between two configurations in C_f is a computationally intractable problem, and can in practise be done only for the simplest of problems. There are methods, many from the area of computational geometry, that guarantee an optimal solution if there is any feasible path, but they are unfortunately very costly if applied to configurations spaces with high dimensionality (more than 3–5 according to [1, 6])

The inapplicability of traditional methods has, during recent years, lead to an exploration of methods where the completeness and optimality demands have been relaxed. One example of this is the probabilistic roadmap algorithm [6], which is neither complete nor optimal. However, in many cases, the paths planned are satisfactory in length. It can also be shown that there is a weaker completeness-property, namely probabilistic completeness. Probabilistic completeness means that the probability of finding a path, if there is one, will approach 1 in limit, if the algorithm is allowed to execute for a sufficiently long period of time.

The probabilistic roadmap algorithm works in two phases: the roadmap construction phase and the query phase. During the first phase, which can be done offline, a roadmap will be constructed that will enable efficient path-planning queries when the robot is online. This roadmap is a graph situated in C_f with nodes and edges that are guaranteed to be collision-free. The graph is created by randomly choosing configurations from C_f and adding these as nodes. The nodes are then connected to each other by means of a local planner for the particular kind of robot that is intended to use the eventual plans. The local planner should very quickly try to find a path connecting two configurations, and here quickness is more important than completeness. If the local planner succeeds, the corresponding edge is added to the graph.

After the roadmap has been generated, it can be used to answer path planning queries. Given a start- and a goal-configuration, the first step is to connect these to the graph. This is done by invoking the same local path planner to find a paths to nodes in the roadmap that are close to the start- and goal-configurations, respectively. If these connections are established, the problem of finding a path is reduced to a standard graph-

Figure 1.2: The relation between the global planner, local planner and the collision checker.

traversal-problem which can be solved using standard search algorithms such as A*.

I will return to the probabilistic roadmap algorithm and its variants in more depth in chapter 4.

1.4 Implementation Framework

As part of this thesis work, a path planning module was developed for the WITAS UAV. In this section, I will give a brief description of the path planner implementation and a rough overview of the WITAS software platform in which the path planner was integrated. I will also provide a description of the environment in which the path planner way tested.

1.4.1 Path Planner Overview

For the implementation of the probabilistic roadmap planner, a general path planning-framework was developed. The framework was designed with the probabilistic roadmap algorithm in mind, but it can also be used for other more or less similar planning algorithms. The task of constructing a path is divided into three fairly independent modules that are depicted in figure 1.2: a global planner, a local planner and a collision checker.

The local planner is constrained by the kinematics of the robot, and has the responsibility of planning a path for the robot between nearby configurations. The local planner generally has to be made specifically for a certain type of robot, in this case a helicopter, and it is the only component that has this connection to the specific robot in question. The reason is that both the kinematics and dynamics of the robot constrain the operation of the local path planner. The other components are generic and can be used for any type of robot.

The path planned by the local planner is sent to a collision-checker in the form of a parameterized curve, in order to check if it intersects with any of the obstacles in the environment.

The role of the global planner is, throughout this report, taken by the probabilistic roadmap algorithm. In a more general setting, the responsibility of the global planner is to patch several of the local paths together into a complete plan taking the robot from a starting position to a goal position. This is done by generating configurations in free space and attempting to connect these by using the local planner. For the probabilistic roadmap algorithm, this is, as has already been explained, done totally at random, but other more deliberative schemes are conceivable.

1.4.2 The WITAS software architecture

In figure 1.3 a schematic overview of part of the WITAS software architecture is shown with some of the major components and the most important connections between these.

A central component of the system is the Task Procedure Execution Module that handles the behaviors of the helicopter. A Task Procedure (TP) can be responsible for executing a particular flight mode of the helicopter, such as monitoring an area, following a vehicle or just flying to a certain way-point. The behavior described by the TP:s can be of varying degrees of complexity where higher level TP:s call TP:s with simpler behavior, e.g. in order to take a picture of a certain place the helicopter has to fly to an appropriate position which is done by the picture taking TP calling a TP for flying to a certain location.

To support the TP:s the WITAS system also includes a wide variety of services. These can be divided into higher level cognitive services and knowledge services which provide more passive knowledge about the world. The path planner developed during this thesis work is an example of a cognitive service.

In the lower part of the figure several modules that act as interfaces to the actual physical helicopter platform are shown. These include the very

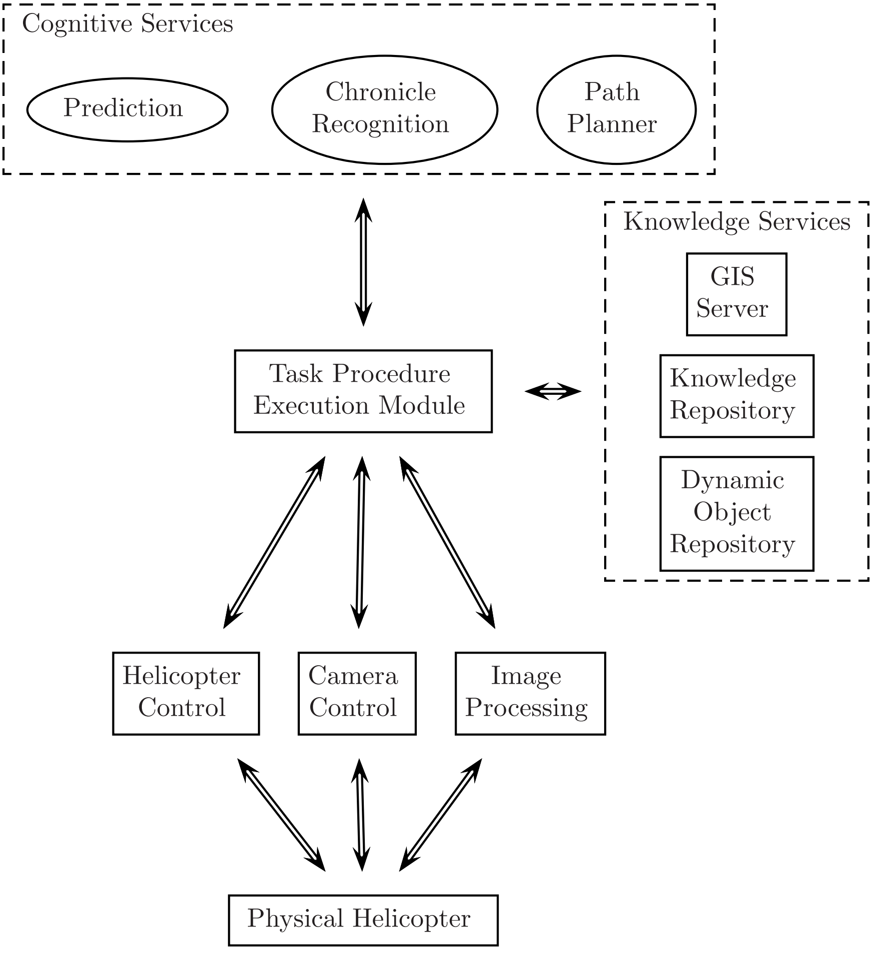

Figure 1.3: The WITAS software architecture

tight control loops where real time behavior is necessary and are controlled by the TP that is currently running.

The implementation is based on a distributed software solution using real-time CORBA. This 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.

The computation hardware used on-board the helicopter consists of three Ethernet-connected PC104 computers running Linux. The lower levels of control system is executed on a P5 266 MHz running RT-Linux to ensure a real time response. The Task Procedure Execution Module and the deliberative services such as the path planner runs on a PIII 700 MHz with TimeSys Linux. The third computer, which is also a PIII 700 MHz, is dedicated to the image processing subsystem and camera control.

In addition to the software that is run on the actual helicopter, there is also an extensive simulation framework used for testing purposes and experiments, complete with a performer-based 3D visualization tool.

1.4.3 World model

Test flights for the WITAS-helicopter are limited to for a few sites in Sweden where the largest one, with most building structures, is the Revinge emergency services training grounds outside Lund (see figure 1.4). A detailed 3D model of the Revinge flight test area has been developed for the project in order to, among other things, be used for simulation of the helicopter and its movements through a realistic environment. The model consists of a number of objects, corresponding to buildings, trees and other structures in Revinge, as well as height information of the underlying terrain. Each of the objects is described as a set of textured polygons.

Since Revinge is rather flat, and most buildings are just a few stories high, this environment is quite simple to negotiate (if the helicopter is not restricted to flying at a very low altitude). In order to create a more challenging world for testing the path planner, a version where all objects are scaled by 20 in the vertical direction, was also used. This provides an environment which is similar to a city center with skyscrapers of reasonable density and height.

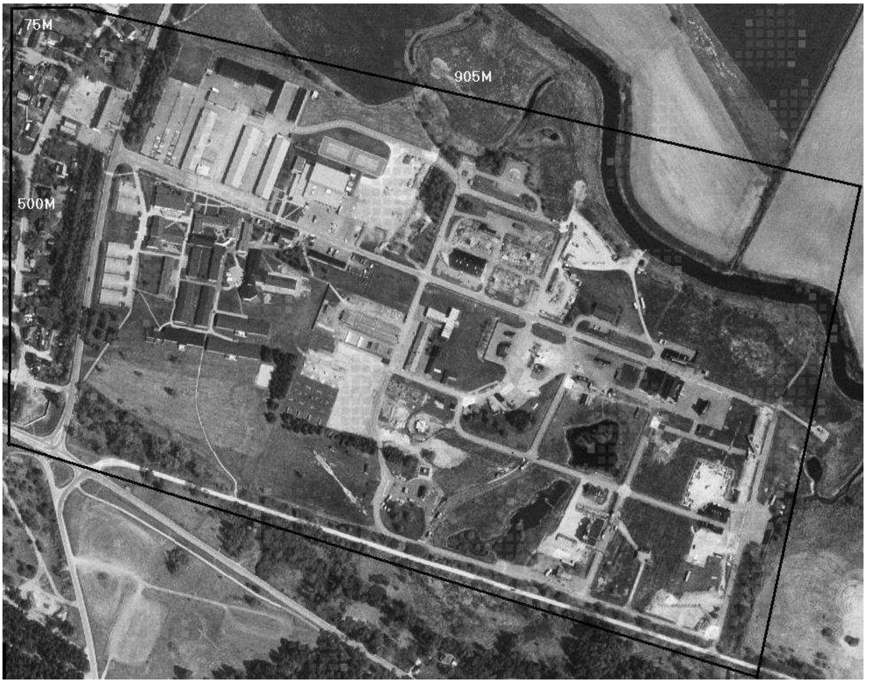

Figure 1.4: An aerial photo over the Revinge rescue worker's training ground.

Chapter 2

Helicopter mechanics and local path planning

While the probabilistic roadmap approach to path planning is a general method, it is still necessary to create an interface between the general algorithm and a specific type of robot. This is the responsibility of the local path planner that connects two given configurations with a path that is possible for the particular robot to execute under its kinematic and dynamic constraints. In this chapter I will describe aspects of helicopter dynamics that are relevant to the planning problem, and derive two alternative mathematical expressions that describe a feasible flight path for the helicopter. In both cases the derivation will result in a parameterized curve, $\mathbf{r}$, in the form of a third degree polynomial with vector coefficients:

$$\mathbf{r}(t) = \mathbf{a}_0 + \mathbf{a}_1 t + \mathbf{a}_2 t^2 + \mathbf{a}_3 t^3 \tag{2.1}$$

2.1 Helicopter kinematics

Disregarding moving parts such as rotors we can view a helicopter as a rigid body in space with 6 degrees of freedom: 3 position coordinates and the 3 angles for orientation (pitch, yaw and roll).

The orientation of the helicopter will however be disregarded for the purpose of path-planning. To see that this is possible we have to take a closer look at the two aspects of path planning where the orientation of the helicopter is involved. The first of these is the responsibility of the path planner to give a full description of the movement of the helicopter, which involves not only its position but also the orientation. The responsibility of determining the orientation of the helicopter can however be delegated to the control-system which is able to automatically align the helicopter appropriately. The pitch and roll during flight depends on the current acceleration of the helicopter, e.g. tilting it forward for acceleration and inward for turns etc, while the yaw is determined by the task the helicopter is to execute, which often is just heading forward for general flight.

The other area where orientation comes into play is for collision detection. This topic will be described in detail in the next chapter, and I will argue that orientation is of little importance, at least in the current applications, due to the roundness of the helicopter (viewing the main rotor as a circular disc) and its relative smallness compared to buildings etc. Of course, if future applications would demand flying in very tight spaces, inside building structures for example, then these assumptions would no longer be valid and the orientation of the helicopter would have to be taken into account.

When flying at normal cruise speed, the helicopter cannot immediately change the direction of flight due to a limited capability for acceleration. This limitation is reflected in a maximal curvature of the flight paths, which introduces a non-holonomic constraint on flight. The constraint will be implemented by adding velocity-information to the helicopter-state and making sure that the acceleration required for flying along a curve at a certain speed never requires more acceleration than what is available.

This non-holonomic constraint is a bit special in that it can be relaxed by reducing the speed of the helicopter or even removed if the helicopter is allowed to stop and acquire a hovering state. Thus the path-planner has the choice of making very sharp turns while negotiating narrow passages if the speed is sufficiently low. For the most part however, when flying through more open space, relatively shallow curves are very much preferred, since they allow for greater speed.

2.2 Current Local Flight Path

For the the purposes of path planning, only the highest level of the control system is of interest. This is the level of acceleration control that specifies a particular acceleration in order to arrive at a certain point in space with a certain velocity. Given the desired acceleration and the current position, velocity, orientation etc of the helicopter, the lower levels of the control system issue appropriate commands to the servos of the helicopter so that the helicopter flies along a path that closely matches the path generated by the local path planner. [16]

The current implementation of the control system separates different parts of the control-task. For the path planner it is mainly the XY-channel and the Z-channel that is of interest since these are the channels that control helicopter movement and position in space.

2.2.1 XY-channel

The navigational controller consists of two specialized control-modes to be used under different circumstances. If the target point does not lie within a 45°-sector measured from the current heading of the helicopter, it will begin by making a sharp turn, roughly in the shape of a circle-segment, until this condition is fulfilled. When the helicopter has reached a heading where the target point lies within this sector, the helicopter switches to a control-mode that will take it along a smooth curve to the target point.

The sharp turn that occurs when the target point lies outside the 45°-sector arises from the turn-mode of the helicopter-controller and lacks a well defined curve-form. Fortunately this part of the controller can be circumvented by restricting the local path planner so that it only connects configurations where the velocity-vectors are within this 45°-sector. This restriction provides a simpler implementation of the local planner, but as will be seen in chapter 4, decreases the density of the graph when used in the probabilistic roadmap algorithm.

The actual curve is described as a third degree polynomial $x \to y$ in the xy-plane where x goes from 0 to 1.

$$y(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \tag{2.2}$$

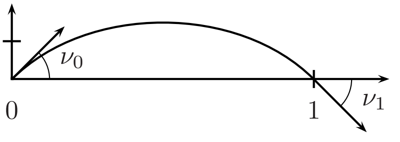

Figure 2.1: The local flight path as seen from above.

Depending on if the the helicopter will stop at the goal point or pass that point in a certain direction, there are two different sets of boundary conditions that will give rise to two slightly different curves.

For the case where the helicopter passes the goal point, the boundary conditions for the polynomial consist of the values in $x = 0, 1$ and the angles between the direction of flight for the helicopter in these positions and the x-axis, ν_0, ν_1 (see figure 2.1). By inserting these boundary conditions into the polynomial at $x = 0, 1$ the coefficients can be found [1]

$$y(0) = a_0 = 0 \tag{2.3}$$
$$y(1) = a_3 + a_2 + a_1 + a_0 = 0 \tag{2.4}$$
$$y'(0) = a_1 = \tan\nu_0 \tag{2.5}$$
$$y'(1) = 3a_3 + 2a_2 + a_1 = \tan\nu_1 \tag{2.6}$$

Solving this equation system and inserting the coefficients in the curve expression gives us

$$y(x) = (\tan\nu_0 + \tan\nu_1)x^3 - (2\tan\nu_0 + \tan\nu_1)x^2 + \tan(\nu_0)x \tag{2.7}$$

In the other case, when the helicopter is supposed to stop at the goal point, the direction condition for this point is replaced by the requirement

[1] The control law used in the control system is derived in a slightly different way, under a small-angle-assumption. That derivation gives $\sin\nu_1$ and $\sin\nu_2$ instead of tan of those angles, and this form is also used in the actual implementation. The effect of choosing sin or tan, is however negligible since they are approximately the same for small angles.

of a zero acceleration at this point:

$$\begin{aligned} y(0) &= a_0 & &= 0 && (2.8)\\ y(1) &= a_3 + a_2 + a_1 + a_0 & &= 0 && (2.9)\\ y'(0) &= a_1 & &= \tan \nu_0 && (2.10)\\ y''(1) &= 6a_3 + 2a_2 & &= 0 && (2.11) \end{aligned}$$

In this case the solution to the equation system gives us the following expression for the path.

$$y(x) = \frac{\tan \nu_0}{2} x^3 - \frac{3 \tan \nu_0}{2} x^2 + \tan(\nu_0) x \tag{2.12}$$

This expression can be parameterized in t by letting $x(t) = t$

$$x(t) = t \tag{2.13}$$

$$y(t) = a_3 x^3(t) + a_2 x^2(t) + a_1 x(t) + a_0 = a_3 t^3 + a_2 t^2 + a_1 t \tag{2.14}$$

or in vector-form:

$$\mathbf{r}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} 0 \\ a_3 \end{pmatrix} t^3 + \begin{pmatrix} 0 \\ a_2 \end{pmatrix} t^2 + \begin{pmatrix} 1 \\ a_1 \end{pmatrix} t = \mathbf{a}_3 t^3 + \mathbf{a}_2 t^2 + \mathbf{a}_1 t \tag{2.15}$$

The parameterized curve is then scaled by d, the distance between the start- and end-position

$$\mathbf{r}_s(t) = d\mathbf{r}(t) = d(\mathbf{a}_3 t^3 + \mathbf{a}_2 t^2 + \mathbf{a}_1 t) = \mathbf{a}_3 d t^3 + \mathbf{a}_2 d t^2 + \mathbf{a}_1 d t \tag{2.16}$$

By multiplying the curve with the rotation-matrix

$$\mathbf{T} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \tag{2.17}$$

the curve is rotated θ degrees where θ is the angle between the x-axis and a line through the start and end-position:

$$\mathbf{r}_{sr}(t) = \mathbf{T}\mathbf{r}_s(t) = \mathbf{T}\mathbf{a}_3 d t^3 + \mathbf{T}\mathbf{a}_2 d t^2 + \mathbf{T}\mathbf{a}_1 d t \tag{2.18}$$

Finally the curve has to be translated to the correct position, beginning at the start-point, $\mathbf{a}_0 = (x_0, y_0)^T$

$$\mathbf{r}_{srt}(t) = \mathbf{r}_{sr}(t) + \mathbf{a}_0 = \mathbf{T}\mathbf{a}_3 d t^3 + \mathbf{T}\mathbf{a}_2 d t^2 + \mathbf{T}\mathbf{a}_1 d t + \mathbf{a}_0 \tag{2.19}$$

2.2.2 Z-channel

There are currently two alternative control-modes for controlling the altitude of the helicopter: one that quickly assumes a certain goal altitude and another one that climbs or descends from one altitude to another along a linear path. For path planning the latter of these is most appropriate since it gives the same behavior regardless of the speed at which the helicopter travels. It also has the advantage of a simpler mathematical form that simplifies collision-checking significantly.

The path traced by the helicopter, using this altitude-controller, is described by

$$z(t) = (z_1 - z_0)t + z_0 \tag{2.20}$$

over the interval $t \in [0, 1]$. Here z_0 is the altitude at the beginning of the path ($t = 0$), while z_1 is the altitude at the end ($t = 1$) of the path.

2.3 Derivation of general three dimensional flight paths

As an alternative to the current flight path a more generalized, true space curve has also been derived. It has the advantage of being able to handle target positions that lie anywhere, not just those restricted to a 45°-sector.

The derivation is done in two steps. First we need to calculate the time, T, for the helicopter to reach the destination point. When T is known we can derive the expression for the parameterized curve, $\mathbf{r}$, describing the movement of helicopter. The estimation of flight time is currently just the time it would take the helicopter to fly between the two points if it would fly along a straight line connecting them at a certain speed.

Let $\mathbf{r}(t)$ be the cubic spline segment parameterized over the time $t = [0, T]$

$$\mathbf{r}(t) = \mathbf{a}_0 + \mathbf{a}_1 t + \mathbf{a}_2 t^2 + \mathbf{a}_3 t^3 \tag{2.21}$$

If $\mathbf{r}(0)$, $\mathbf{r}'(0)$, $\mathbf{r}(T)$ and $\mathbf{r}'(T)$ are known, the coefficients can be determined in the following manner according to [12]

$$\mathbf{a}_0 = \mathbf{r}(0) \tag{2.22}$$

$$\mathbf{a}_1 = \dot{\mathbf{r}}(0) \tag{2.23}$$

$$\mathbf{a}_2 = 3\frac{\mathbf{r}(T) - \mathbf{r}(0)}{T^2} - 2\frac{\dot{\mathbf{r}}(0)}{T} - \frac{\dot{\mathbf{r}}(T)}{T} \tag{2.24}$$

$$\mathbf{a}_3 = 2\frac{\mathbf{r}(0) - \mathbf{r}(T)}{T^3} + \frac{\dot{\mathbf{r}}(0)}{T^2} + \frac{\dot{\mathbf{r}}(T)}{T^2} \tag{2.25}$$

The directional part of the vectors $\dot{\mathbf{r}}(0)$ and $\dot{\mathbf{r}}(T)$ is determined by the direction of flight for the helicopter at the beginning and the end of the flight path, while the magnitude can be chosen to give an appropriate curvature, or just be left at unit-length.

2.4 Acceleration Controller

Having generated the curve that the helicopter is supposed to follow, there must be a corresponding control-mode that makes the helicopter fly along the given path. An analysis of the control system and issues regarding stability etc. are beyond the scope of this report, but I will briefly describe how the acceleration of the helicopter is determined from the flight path.

2.4.1 Current Controller

The local path planners described in the two preceding sections both generates a flight path described by a third degree polynomial:

$$\mathbf{r}(t) = \mathbf{a}_0 + \mathbf{a}_1 t + \mathbf{a}_2 t^2 + \mathbf{a}_3 t^3 \tag{2.26}$$

During runtime, the flight path is generated for each time step by the control system, so in reality a new curve is generated repeatedly. Due to boundary conditions and the nature of the curve-construction, the new curve will in general follow the previous one very closely.

To calculate the acceleration required at each point along the curve, equation 2.26 has to be derived twice:

$$\ddot{\mathbf{r}}(t) = 2\mathbf{a}_2 + 6\mathbf{a}_3 t \tag{2.27}$$

	$-45°$	$-30°$	$-15°$	$0°$	$15°$	$30°$	$45°$
$-45°$	2.43	1.16	1.74	2.45	3.38	4.83	7.66
$-30°$	2.83	0.84	0.35	0.63	1.18	2.40	5.14
$-15°$	3.24	1.04	0.18	0.12	0.52	1.67	4.26
$0°$	3.70	1.30	0.28	0.00	0.28	1.30	3.70
$15°$	4.26	1.67	0.52	0.12	0.18	1.04	3.24
$30°$	5.14	2.40	1.18	0.63	0.35	0.84	2.83
$45°$	7.66	4.83	3.38	2.45	1.74	1.16	2.43

Table 2.1: Maximal deviation in percent of the distances between start- and end-points for varying headings at these points

and since it is a new curve for each time-step, we are only interested in what to do at the beginning of the curve, i.e. for $t = 0$

$$\mathbf{a} = \ddot{\mathbf{r}}(0) = 2\mathbf{a}_2 \tag{2.28}$$

The analytical curve derived by the algorithm is however not identical to the iterated curve generated by the controller. This discrepancy makes this type of controller a bit problematic for the issue of collision checking, since it may result in a collision during runtime which would go undetected during compilation of the roadmap.

For a quantitative estimate of this discrepancy, a series of tests were made where the initial curves where compared to curves formed by iteratively traveling in small steps along the curve and recalculating the curve for each step using the technique described above. These tests were made for a series of different angles for the start- and end-points and the results are listed in table 2.1. As can be seen, the deviation is negligible for small angles but starts to make an impact when one or both of the angles reaches 30°or more. The largest deviation occurs when the angles are 45°with opposite sign, and this case is shown in figure 2.2.

Naturally the risk of having the helicopter colliding with obstacles in the environment during flight is totally unacceptable, and the largest possible deviation between the planned and flown curve has to be added to the safety margins if this control-mode is to be used with the physical helicopter.

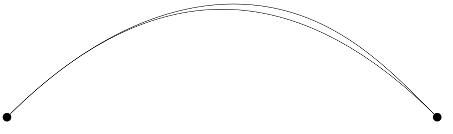

Figure 2.2: The maximal deviation between planned path and the actual flight path ($\nu_1 = 45°$ and $\nu_2 = -45°$)

2.4.2 Path following controller

A better solution, which may avoid the problem above, might be to change to a path-following controller that instead of just flying to a certain point, follows a given curve. The helicopter controller already incorporates a controller-mode whose task is to track a moving object, e.g. a ground vehicle, by locking on to a point moving in accordance with this object[16]. This controller could be used to guide the helicopter along the parameterized curve by having the path planner provide a point moving along the curve and having the helicopter track this point. This approach has not yet been tested but provides an interesting topic for future investigations.

Unlike the current choice of flight path and controller, this approach should make the path flown by the helicopter identical to the planned path in theory. In practice, different kinds of noise like wind-gusts etc, will also come into play resulting in some deviation from the analytical path.

Another advantage to this type of controller is that it also makes an effort to return to the planned path if it for some reason has drifted away from it. The aim of the current controller is simply to reach the destination point with the correct velocity vector, and it chooses the best trajectory from its current state. This means that if the helicopter with the current controller is outside its correct path, it will make a new path, which will not have been checked for collision. If the new curve happens to intersect any obstacle, this will of course have disastrous consequences[2].

[2]In recent work, a trajectory following control mode has been added to the suite of helicopter control modes in the project. This control mode can be used to follow the curves generated by the path planner more accurately.

Chapter 3

Collision Detection

The construction of a flight path for the helicopter that was described in chapter 2 does not in any way rule out potential collisions with objects that might lie along the proposed route. Each planned path must therefore be checked so that, during the flight along a path, no parts of the helicopter intersect any of the other objects in the world. It is only after this check has been made that the generated path can be considered a safe flight path and be included as part of the roadmap.

In this chapter I will present a collision checker, designed to check if a curve in the form derived in the previous chapter intersects the polygonal model of the world. I will also provide a rough estimate of the computational cost for the collision checker since its performance is of great importance for the overall performance of the path planner.

3.1 Previous work

The collision checker used in this project is based on a recursive decomposition of the world model where each part is bounded by a box. In this section I will present the original algorithm before moving on to the modifications that was made for this project.

3.1.1 Bounding Volumes

A common method to simplify and lower the computational cost for collision checking between complex models is to place a bounding volume around each object and test these for intersections. These bounding volumes generally have a much simpler geometry than the actual objects in the world, which allows for fast and simple implementations. The downside is that false positives are introduced, i.e. the collision checker might report a collision even if there is none. This does not in any way make bounding volumes useless, since it provides the possibility of quickly rejecting potential collisions between objects that don't happen to be very close together. In reality, it is quite common that an absolute majority of the objects tested are well separated and can be easily rejected using bounding volumes. Many collision detectors therefore incorporate this method for quick rejects and use some more advanced methods for the trickier cases.

Bounding volumes can be constructed in several ways at different levels of complexity. The simplest shapes are axis aligned bounding boxes (AABB) and spheres. Both these types are simple to generate, especially AABB:s where the bounds are just the maximum and minimum of the points contained along each axis. They are both relatively easy to check for collisions.

However, spheres and AABB:s fit long and thin objects poorly (for objects not aligned along the coordinate-axes in the AABB-case). This can be overcome by using oriented bounding boxes (OBB) that are aligned along the objects in a proper manner.

The choice of alignment is not trivial. One way of doing this for models made out of polygons is presented in [5]: The OBB is aligned so that two of its axes are in the directions of the largest and smallest variation of the vertices of the polygons belonging to the object while the third axis is chosen perpendicular to the two others. These axes are found by means of principal component analysis (PCA): If $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are the vertices of the polygons, $\boldsymbol{\mu}$ is given by

$$\boldsymbol{\mu} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \tag{3.1}$$

where the covariance matrix $\mathbf{C}$ is given by

$$\mathbf{C} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T \tag{3.2}$$

The directions of maximum and minimum variation are now simply the largest and smallest eigenvectors of $\mathbf{C}$, so these eigenvectors can be used for aligning the OBB.

3.1.2 Hierarchical Decomposition of OBB:s

Instead of checking for a potential collision with each and every one of the objects in the world, a divide-and-conquer-strategy for the collision problem can be applied [5]. This algorithm works by first bounding all the polygons in the model with an OBB. The OBB is then split into two sub boxes and the polygons are divided into two groups depending on which half of the box their center-points happen to lie in. A new OBB is then created for each of the two groups, bounding the polygons in the corresponding group. The splitting of the bounding boxes is done recursively all the way down to individual polygons, or until it is no longer possible to separate the polygons into different groups. The cut of the box into two halves is, if possible, done along its longest axis, or otherwise along its 2nd or 3rd longest.

3.2 Implementation

For the purpose of path planning, when we do collision checking, we are not solely interested in checking whether two objects intersect each other. The question is rather if one of the objects, the robot, will intersect the other during the movement of the robot along a certain path. The volume of space traced out by a 3D-model of the helicopter traveling along a non-linear curve in space is quite complex, with both curved edges and non-flat faces.

Fortunately it is not necessary to use a full 3D-model of the helicopter in most cases. Since the helicopter is a relatively small object compared

Algorithm 1 Collision checking: path–OBB-tree

```
function intersects(p, n)
  if intersects(p, obb(n)) then
    if leaf(n) then
      return true;
    else
      return intersects(p, leftsubtree(n)) ∨ intersects(p, rightsubtree(n))
    end if
  else
    return false
  end if
end
```

to the environment in which it operates, and doesn't deviate very much from a spherical object, it can instead be seen as a simple point-object, if its maximal radius is added to the safety margin placed around the static obstacles. It is only for precise flight through very narrow passages, that the advantages of a more detailed 3D-model of the helicopter could be utilized. As a point-object, it will trace a simple curve through space, and it is enough to check if this curve intersect any other object in the world.

Checking for intersections is done with a modification of the OBBTree-algorithm described in section 3.1.2. An OBB-tree is built for the world-model during the initialization of the program in the same way as in the algorithm described in section 3.1.2, but instead of checking for a collision between two OBB-trees, the collision-checker tests if a parameterized space curve intersects the OBB:s of the tree.

When a 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 path since the whole model is within this OBB. On the other hand, if there is a collision, the two sub-OBB-trees have to be checked. This process continues recursively down to the leaves of the OBB-tree. If a collision is reported for any of these leaves, a collision has occurred. The function intersection-test function for a cubic curve and an OBB is shown

Algorithm 2 Collision checking: path–OBB

```
function intersects(p, b)
  if start(p) ∈ b then
    return true
  end if
  for i = 1 to 3 do
    for all (p ∩ lowerplane(b, i)) ∪ (p ∩ upperplane(b, i)) do
      c ← true
      for j = 1 to 3 do
        if i ≠ j then
          if below p, lowerplane(b, i) ∧ above p, upperplane(b, i)∧ then
            c ← false
            break
          end if
        end if
      end for
      if c then
        return true
      end if
    end for
  end for
  return false
end
```

more formally in algorithm 1.

The check for collisions between a curve and an individual OBB is shown in algorithm 2 and is done by first localizing the points where the polynomial crosses the planes that bound the box. If any of these points lie on the corresponding side of the box, then a collision has occurred.

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

$$\mathbf{r}(t) = \mathbf{a}_0 + \mathbf{a}_1 t + \mathbf{a}_2 t^2 + \mathbf{a}_3 t^3 \tag{3.3}$$

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:

$$a_0 + a_1 t + a_2 t^2 + a_3 t^3, \quad a_i = \mathbf{a}_i \cdot \hat{\mathbf{n}} \tag{3.4}$$

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 t for which $\mathbf{r}(t)$ intersects the plane is given by

$$a_0 + a_1 t + a_2 t^2 + a_3 t^3 = d \tag{3.5}$$

The solution to this general cubic equation is given in detail in appendix A.1. Inserting the solutions back into equation 3.3 results in one or three points on the plane. As a last step, these points are compared to the planes corresponding to the four neighboring faces of the OBB. If they all lie outside these planes, no collision has occurred, while if any of them lies within the space confined by these planes, a collision has occurred.

These calculations are currently done for each of the six faces of the OBB, or until a collision is detected.

3.3 Estimation of computational cost for collision checking

Since each invocation of the local path planner will be followed by a test if the path is collision-free, the collision checking algorithm determines a large part of the computational cost of this approach. It is therefore important to compute an estimate of how this cost scales up for larger or more detailed models.

To provide an exact analysis of this cost is of course very complicated and outside the specification of the original thesis problem. In any case, I will provide a rough estimate in this section and save a formal complexity analysis for future work. The following assumptions will be used:

Uniform model The model will be assumed to consist of evenly spread and sized polygons. This is actually a worst-case scenario as will be argued shortly.

Straight curves The flight curves are assumed to be straight. This assumption simplifies the analysis and should not have any large impact on the results

3.3.1 Construction of OBB-tree

The OBB-tree is a binary tree which gives it a logarithmic depth in the number of polygons, assuming that it is well-balanced. This assumption is equivalent to the assumption of an even distribution of polygons in the model.

For each node in the tree, the construction-algorithm has to calculate the covariance-matrix, $\mathbf{C}$, create the OBB, and divide the polygons into the two sub-trees. All these operations are done in linear time. Looking at the tree as a whole, each polygon will be processed exactly once on each level in the tree. Thus the complexity of the tree-construction would be $O(n \log n)$ with regard to the number of polygons.

With the present model of Revinge, the tree-construction takes less than a minute and stands for a smaller part of the total time for roadmap-construction. Due to the low complexity of $O(n \log n)$, the time consumption will not become a major problem until the model is several magnitudes larger.

A more important limiting factor is the amount of memory that is available for holding the finished OBB-tree. The memory requirement is not only a factor that is important during initialization but during the whole course of the program execution. Being used on board the helicopter, together with all other programs needed and utilizing the same memory resources, it is very important that the OBB-tree only uses a fairly low amount of memory. With the current implementation the tree already takes up around 40 MB of memory for the current world-model, and in order to assure quick execution of the collision checker, the tree has to reside in the primary memory of the computer during path planning. Since the size of the tree increases linearly in the number of polygons, even a slightly larger model, of say a couple of times the number of polygons would begin to become too much memory to assign just to the path planner.

For models where the increased number of polygons is a reflection of a higher degree of details, the memory usage could be limited to some degree by introducing a cut-off threshold when reaching sufficiently small OBB:s. There is also still some room for micro-optimization of the code by e.g. using a more memory-efficient matrix-representation.

Since the time complexity is $O(n \log n)$ while the space complexity is $O(n)$, only the cost of building the tree would increase if the tree would be unbalanced due to the increased depth. This is a fortunate situation since it is the memory consumption that is the most problematic area.

3.3.2 Recursive collision checking with the OBB-tree

When checking for a collision between a curve and an OBB-tree, each OBB that intersects the curve will be checked once. The number of OBB:s that are intersected will statistically depend on the length of the curve in comparison to the density of polygons and their size. I will start by investigating the cost for the two extreme cases: a curve shorter than the minimal OBB:s and a curve traversing the whole OBB-tree, and then combine these two analyses for the general case.

If the curve is very short, it can easily be seen that for each step of the recursion down the OBB-tree, the curve will generally only intersect or be contained in the OBB corresponding to one of the subtrees. Thus the algorithm will only have to investigate one branch of the tree and the complexity would be $\log n$ in the number of nodes. If the polygons are unevenly spread, the situation would actually improve, since that would leave gaps which makes it possible to break off the search at an earlier stage. Of course the increased depth of the tree would mean that some searches would have to go deeper, but this addition is not enough to cancel out the positive effect. This is so because it is much more likely for the path to intersect a large polygon higher up in the tree than a smaller one deep down in the tree.

For a long curve, traversing the whole model, we look at the OBB:s corresponding to the leaf-nodes. The curve will in this case intersect a series of OBB:s throughout the model. If the OBB:s are evenly spread in the model, and the model is approximately cubic, there will be about $\sqrt[3]{n}$ OBB:s in each dimension. Since the curve is one dimensional and of the same length as the diameter of the model, it will intersect roughly $\sqrt[3]{n}$ OBB:s, and this would therefore also be the cost in this case. For a flat model like the one for Revinge and most other models of open terrain, the cost would, by the same argument become approximately $\sqrt{n}$. Again, open areas in the model might reduce how deep the algorithm has to descend the tree, and the same argument as above applies here too.

Bringing these two analyses together, we can compare the cost of having a few very long paths in relation to having many short paths. To get from one end of the graph to the other using just one path, the cost for checking this path for collision would be $\sqrt[3]{n}$ or $\sqrt{n}$ as described above. If we instead use path lengths that are of the same general size as the smallest OBB:s in the tree, the first analysis applies and the cost to check each one of these would be around $\log n$. To get from one end to the other would require $\sqrt[3]{n}$ paths of this length patched together, giving a total cost of $\sqrt[3]{n} \log n$. This is not much more than checking just one long path but gives two important advantages:

1. Several shorter paths provides many connection points along the path where other paths can be connected.
2. It is more likely that a short path does not collide with any obstacle, which provides more edges in the roadmap.

There is however nothing gained in shortening the paths further, since the cost for collision checking would still be $\log n$ for each path.

3.3.3 Cost for curve–OBB intersection test

Table 3.1 presents the number of operations necessary to check for an intersection between a parameterized cubic curve and an OBB. The average cost is calculated by noting how many times in mean the different steps of the collision checking algorithm are executed for 1000 curves in the Revinge model.

As can be seen in the table, this is about 4-5 times more than the 180 operations, worst case, or 3-4 times more than the 100 operation average-case used by the OBB-OBB-intersection algorithm used in [5]. The approach taken in this report has however the advantage of testing the whole path against the OBB in one go, while the use of an algorithm for testing two OBB:s against each other in many cases would require several tests, either

Calculation	Mul	Add	Func	n worst-case	n average
Projection on axis	12	8	0	3	2.96
Cubic equation (D > 0)	17	9	3	0	1.81
Cubic equation (D < 0)	26	10	4	6	1.12
Point-comparison	3	2	0	72	33.36
Sum (worst case)	408	228	24		
Sum (average)	195	118	10		

Table 3.1: Cost for intersection test between curve and OBB (calculated in appendix A.2). Mul and Add stands for the number of multiplications and addition respectively, while Func stands for the number of calls to mathematical functions like roots and trigonometric functions.

by constructing an OBB-tree around the whole path or by using some iterative method of moving an OBB wrapping the helicopter along the path and checking for collision several times along the path.

Chapter 4

Probabilistic roadmaps

The probabilistic roadmap algorithm was introduced in chapter 1.3 and in this chapter I will go into more depth, describing the basic algorithm in detail together with some extensions. I will also describe how the modified algorithm has been applied within the WITAS-project. Finally I will show some tests comparing how well different approaches of the probabilistic roadmap algorithm perform for the particular path planning problem specified in the project.

4.1 Theory

The probabilistic roadmap algorithm has been the subject of intensive research during the last decade and several variants of the algorithm have been developed and described in the literature. In this section, I will begin by presenting the common core of the probabilistic roadmap algorithm as it is presented in a number of research papers [6–11, 17, 18], and give the bare minimum for a working implementation. Most of the papers propose different kinds of extensions to improve on the basic algorithm. I will present two of these that have been tested within this project in the following two subsections.

4.1.1 Basic Algorithm

The Probabilistic roadmap algorithm is a general path planning algorithm applicable to a wide variety of problems. However, in order to plan a path for a specific robot it requires a domain specific local path planner that can produce a simple plan for taking the robot from one configuration to another nearby one. The local planner is called repeatedly by the main algorithm and contributes in many cases to most of the computational cost of the algorithm. It is therefore important that it is able to generate paths fairly quickly. However, it is not necessary for the local planner to actually be able to find a path under all circumstances, even if there is one, since the main algorithm usually can find alternative routes. A simple example of a local planner, suitable for many robots, is a planner that just connects the configurations with a straight line in configuration space if that is possible or fail if there are any obstacles along the line.

The path planning process can be divided to two parts: an offline preparatory phase and an online query-phase. The preparation-phase consists of the generation of a graph that represents the free space as closely as possible. This graph is produced by randomly choosing n configurations in the free space and letting these belong to the set N of nodes in the graph. Nodes lying close together, say within distance r according to some metric d for the configuration space, are then checked for possible connections by invoking a domain specific local planner. If a connection can be made, an edge between the nodes is added to the set E of edges in the graph. The algorithm for graph-generation (Algorithm 3) is shown in more detail on the next page, and an example of a generated graph using linear segments as local paths is shown in figure 4.1.

The actual planning of a path between the current and the goal configurations of the robot is done in the query-phase. It begins by first connecting the robots current configuration and the goal configuration to nearby configurations in the graph, using the local planner. After the start- and goal-nodes are connected to the original roadmap, the problem is reduced to a graph traversal problem that can be solved by standard search algorithms such as A* (see e.g. [13]). If no node in the graph is directly reachable from the start-configuration by means of the local planner, it might still be possible to find a path by extending the graph in the neighborhood of the

Algorithm 3 Probabilistic roadmap construction — basic algorithm

$N \leftarrow \emptyset$
$E \leftarrow \emptyset$
for $i = 0$ to n **do**
  $c \leftarrow$ random free configuration
  $N_c \leftarrow \{x \mid d(c,x) < r\}$
  $N \leftarrow N \cup \{c\}$
  for all $n \in N_c$ in order of increasing $d(c,n)$ **do**
    if connectable(c,n) **then**
      $E \leftarrow E \cup \{\langle c,n\rangle\}$
    end if
  end for
end for

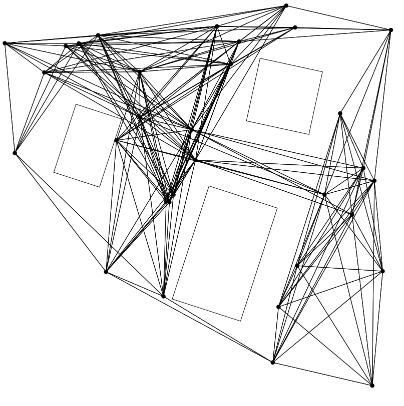

Figure 4.1: Example of probabilistic roadmap using a straight line local path planner.

start-position. This extension is made by randomly adding new nodes close to the start position and attempting to connect them both to the graph and the start-configuration. If this fails (after a certain numbers of tries), the planner fails and the the start and goal configurations are considered impossible to connect. The same method can of course be applied to the goal-configuration if that configuration is not reachable from the graph.

After a path has been found it is usually subjected to some kind of smoothing-procedure, in order to even out the unnecessary jaggedness of the path that often occurs as a result of the probabilistic nature of the algorithm. One way of doing this is by iteratively move configurations closer and closer to a straight line, while repeatedly checking that the new path is acceptable to the local path planner.

4.1.2 Visibility Based Probabilistic Roadmaps

The basic roadmap-algorithm generally produces graphs that are very dense in open areas while the more difficult areas get a much sparser coverage. This is a very much unwanted effect and in order to solve this problem an extension called Visibility-based Probabilistic Roadmaps was proposed in [7, 17, 18].

In this algorithm two types of nodes in the graph are distinguished: guards and connections. The guards are the nodes that are intended to cover as much as possible of free space 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 that part of free space that is reachable from the guard in question using the local path planner.

As in the basic algorithm, the visibility-based roadmap is constructed by repeatedly choosing 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 which depend 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 guards belonging to a single connected component (or just one guard) can be reached. Since this configuration already lies in the visibility domain of some guard and it doesn't improve the connectedness of the graph, it is disregarded.

A more detailed description of the algorithm (Algorithm 4) is shown on the facing page.

4.1.3 Multi-level planning for non-holonomic robots

Non-holonomic constraints restrict the robot to a more limited choice of movements through the configuration space which makes it harder to connect two robot-configurations. In many cases it also complicates the local path planner which leads to a higher cost for each invocation. Taken together this significantly increases the computational demands when using probabilistic roadmaps to capture the robots free-space.

In [14], a method is proposed where some or all of the non-holonomic constraints are lifted during the roadmap construction phase. In many cases this leads to a much simpler and faster local planner but naturally also generates paths that will not directly be negotiable for the non-holonomic robot. There are however often paths for which the non-holonomic constraints do hold, that lie close to these preliminary paths. During querying, this simplified roadmap can therefore be used to find a first approximate path for the robot.

The preliminary path can then be transformed into a path that meets the non-holonomic constraints by using a local variant of the probabilistic roadmap algorithm. If the holonomic path was planned with some clearance to any obstacles, then configurations within this free area can be added randomly close to the preliminary path. These configurations can then be connected with a local planner that takes the non-holonomic constraints into account. Since it is already certain that there is a "corridor" of some sort between the two configurations, a path can be found very quickly. In many cases the initial and end-configuration of a path-segment can even be connected directly in open areas.

Algorithm 4 Visibility-based probabilistic roadmap construction

```
G ← ∅                                        /* guards           */
C ← ∅                                        /* connections      */
E ← ∅                                        /* edges            */
n ← 0                                        /* number of tries  */
while n < M do
  q ← random free configuration
  g_vis ← ∅; G_vis ← ∅
  for all components G_i of G do
    for all nodes g of G_i do
      if q ∈ Vis(g) then
        if g_vis = ∅ then
          g_vis ← g; G_vis ← G_i
          break
        else
          C ← N ∪ {q}
          E ← E ∪ {⟨q, g⟩} ∪ {⟨q, g_vis⟩}
          Merge components G_i and G_vis
          break
        end if
      end if
    end for
  end for
  if g_vis = ∅ then
    G ← G ∪ {q}; n ← 0
  else
    n ← n + 1
  end if
end while
```

4.2 Probabilistic Roadmaps in the Helicopter Domain

Both the visibility-based algorithm and the multi-level method for non-holonomic robots were tested in the Revinge operational environment used in the WITAS UAV Project, as well as a straight forward implementation of the basic algorithm.

4.2.1 Adaption to helicopter domain

As described in section 2.1, a helicopter configuration consists of a three-dimensional position vector and a three-dimensional velocity vector. Each node in the roadmap holds a certain helicopter-configuration and will therefore specify these 6 parameters.

The existence of a velocity vector at each node in the graph means that the helicopter can only approach the node from a certain angle and has to leave it from the opposite side. This implies that we have to use a directed graph, since an edge entering a node must be handled differently from one leaving the node.

The use of a directed graph also provides a greater freedom in choosing a local path planner for the helicopter which can be useful if the movement of the robot is asymmetric in some sense. This is actually the case with the Yamaha RMAX helicopter currently used as the actual physical platform, where the main rotor creates an air cushion which limits the speed downwards while the upward speed can be significantly higher. However, the current control system does not utilize this asymmetry but rather works within the same limit of speed in both ascent and descent [15].

4.2.2 Basic algorithm

The basic probabilistic roadmap-algorithm has rather few requirements and it is a relatively straightforward procedure to use it in the helicopter-domain. The only change required when using it with a directed graph is that when a new node is added, it should be checked for connections both to and from the other nodes in the graph. The modified algorithm is shown on the next page (algorithm 5).

Algorithm 5 Probabilistic roadmap construction — basic algorithm for directed graphs

$N \leftarrow \emptyset$
$E \leftarrow \emptyset$
for $i = 0$ to n **do**
  $c \leftarrow$ random free configuration
  $N_c \leftarrow \{x \mid d(c, x) < r\}$
  $N \leftarrow N \cup \{c\}$
  for all $n \in N_c$ in order of increasing $d(c, n)$ **do**
    if connectable(c, n) **then**
      $E \leftarrow E \cup \{\langle c, n\rangle\}$
    end if
    if connectable(n, c) **then**
      $E \leftarrow E \cup \{\langle n, c\rangle\}$
    end if
  end for
end for

Some papers, e.g. [10], stress the importance of using acyclic graphs. This is in my opinion doubtful for at least some applications as argued below, even if the graph is undirected. For directed graphs the avoidance of cycles would create an even worse problem since there would be a lot of nodes where it is only possible to arrive at them but not leave them, and vice versa. The presence of cycles is in any case, not really a problem since there are many good graph-traversal algorithms that are not affected by cycles, e.g. A*.

Permitting cyclic graphs also introduces another advantage in often generating shorter flight paths which is also noted in [10] even though it is proposed that post-processing of the finished plan is sufficient. The importance of allowing cycles might not be that great for the articulated robots that are the main application studied in [10] where finding any path is of most concern. On the other hand, for a free space with many "holes", e.g. low altitude helicopter flight between high buildings, it is crucial that long detours are avoided, since the path usually doesn't "jump" over the holes in free space during smoothing.

4.2.3 Visibility-based algorithm

The necessity of using directed graphs affects the visibility-based roadmap algorithm to a larger extent than what is the case for the basic algorithm. A basic concept for this algorithm is that of a connected component which is a set of guards that are mutually reachable from each other. This is a straightforward concept for undirected graphs but it turns out that the straightforward extension to directed graphs, using the same definition isn't totally appropriate for use with the visibility-based roadmap algorithm.

Looking at algorithm 4, components are used to avoid adding unnecessary connections between guards that already are connected. Directed graphs have a more complex structure and we can no longer use this quick and simple method for checking if two nodes are graph connected. Instead, the more direct approach of applying a graph search is used in this implementation and a new connection is added only if the destination-guard cannot already be reached from the source-guard. The computational cost of doing this search which is nicely circumvented by using components in [7, 17, 18], is not that great. In the current implementation, some basic profiling tests show that it is responsible for just a few percent of the total computation time.

An example of how to implement the variant for directed graphs is provided as algorithm 6. A small modification of the algorithm that proved to increase the efficiency of it is to connect new connection-nodes to every node that can be reached. This of course leads to a higher number of edges in the graph, which results in a somewhat higher use of memory, but the extra edges pay off by giving drastically shorter routes in the graph.

4.2.4 Multi-level planning

Path planning for the helicopter lends itself very well to the multi-level approach for non-holonomic robots described in section 4.1.3. Due to the hover-capability of the helicopter, the non-holonomic constraints can be lifted as was mentioned in section 2.1 and the helicopter is therefore able to travel any path, albeit slowly. This simplifies the transition step from the preliminary path, only respecting holonomic constraints, to the final one that also respects the non-holonomic constraints since the planner can

Algorithm 6 Visibility-based probabilistic roadmap construction for directed graphs

```
G ← ∅                                           /* guards */
C ← ∅                                           /* connections */
E ← ∅                                           /* edges */
n ← 0                                           /* number of tries */
while n < M do
  q ← random free configuration
  g_to ← ∅; G_vis ← ∅
  for all nodes g of G do
    if q ∈ Vis(g) then
      g_to ← g
      break
    end if
  end for
  if g_to ≠ ∅ then
    for all nodes g of G do
      if g ∈ Vis(q) ∧ ¬connected(g, g_to) then
        C ← N ∪ {q}
        E ← E ∪ {⟨g, q⟩} ∪ {⟨q, g_to⟩}        /* or E ← E
                                                ∪ {⟨x, q⟩ | connectable(x, q)}
                                                ∪ {⟨q, x⟩ | connectable(q, x)}
                                                */
        break
      end if
    end for
  end if
  if g_to = ∅ then
    G ← G ∪ {q}; n ← 0
  else
    n ← n + 1
  end if
end while
```

always reduce speed and thereby bring the path arbitrarily close to the preliminary path.

For the implementation of the holonomic local planner, a straight line-planner was chosen, because of its simplicity. The transition step was also done in a simple manner: A velocity vector was chosen for each node in the planned path, aligned parallel with a line drawn between the two neighboring nodes. If the path planned by the normal local planner through this node with the assigned velocity vector is collision free, the straight line segment will be replaced by the new curved path. If not, the straight line segment will be left in place and the helicopter is required to hover at its endpoints, which makes it possible for the helicopter to travel in a straight line between these two points.

Some tests were made to see how often the helicopter would have to slow down to a hovering state with this simple transition method. The tests were done by letting the helicopter travel through the model of Revinge with high houses along different paths. Each path consisted of around 10-15 nodes and for flights through the more open areas of Revinge the helicopter had to stop and hover in 2 of these on average. When flying through the tight areas with building in the north west of Revinge, the number of hover points rose to 4, while for flight through open air, the helicopter never needed to stop. A more detailed empirical study of this technique would have to be made in order to verify the tests made here but the initial evidence is promising.

4.3 Smoothing of flight route

The path produced by the probabilistic roadmap algorithm is in many cases not as straight and smooth as was hoped for. Therefore it is useful to do some post-processing of the generated path in order to smooth out the worst bumps and unnecessary detours. In the current implementation of the path planner a very simple, but still quite effective, procedure is used. The procedure iterates over the nodes of the path and for each node it tries to place it just in the middle between its two neighbors and adjust the velocity vector so that it is parallel with the line between the two neighbors.

Algorithm	Nodes	Edges	Time	Memory	Coverage	Length
Basic	8 000	115 378	9.57	59MB	68%	837 m
Visibility	4 997	10 150	19.36	51MB	65%	839 m
Linear	5 000	245 702	14.39	61MB	97%	816 m

Table 4.1: Comparison between different variants of the roadmap-algorithm.

4.4 Comparison of roadmap variants

In order to see how some of the different variants of roadmap-algorithms perform in the helicopter domain, a series of tests were performed. For these tests the model of Revinge with the houses scaled in height were used to provide a more challenging environment. The results of these tests are provided in table 4.1. The size of the graphs where chosen with the purpose of showing the strength and weaknesses of the different algorithm.

As can be seen, the use of a linear local path planner as a first stage provides a much better result than planning directly for the helicopter path. The main reason for this is that the linear planner can connect with another node in any direction and is not just limited to nodes within a 45° sector. This is also why the linear planner gives so many edges in the graph even if it has fewer nodes than the basic algorithm. This is not only positive since it increases the memory requirement and slows down the run time search through the graph.

The low number of edges is the main advantage of the visibility based roadmap algorithm. As can be seen in the table, it provides almost the same coverage and path length as the basic algorithm even if it has only 10% as many edges and just a bit over half the number of nodes as the basic algorithm in this test. Unfortunately this reduction in memory usage comes with a longer time required to build the roadmap.

Chapter 5

Discussion

5.1 Future directions

5.1.1 Flight along more general curves

One of the most pressing problems to investigate is a closer integration of the path-planner with the helicopter control system. The main objective for the present control-system is to bring the helicopter to a certain point in space (which is in itself a hard problem to solve) rather than restricting the helicopter to follow a certain, well-defined curve. In section 2.4.2 an alternative solution was briefly presented, which uses an existing control-mode for locking on a moving point. The use of this mode would provide greater freedom to use a wide variety of different types of flight paths. It should also provide more accurate tracking of the specified path since the helicopter would strive to return to the path whenever it deviates from it rather than as with the present controller, just plot a new optimal curve to the goal.

Another advantage when using this type of controller is that a more general class of flight curves can be used since the path planner could produce any series of configurations as long as they do not violate the constraints placed on the helicopter movements. Allowing a freer choice of flight curves would however require a more general collision-checker. However, the current collision-checker requires that it is possible to solve the intersection of the curve and a plane, preferable analytically. An alternative means of checking if the curve intersects the OBB-tree would be to wrap the curve into an OBB-tree as well and check the two OBB-trees for intersection. This would imply a greater number of OBB–OBB intersection tests, but since these tests can be done faster than the curve–OBB intersection test, there might be no significant speed-loss.

5.1.2 Integration of visibility-based and non-holonomic roadmaps

The preliminary investigation of the method in [14] shows great promise with a much denser connectivity of the roadmap and is well worth investigating further. It would also be interesting to try and combine this method with visibility-based probabilistic roadmaps, especially since these are designed primarily for undirected graphs.

5.1.3 Incremental roadmap-construction

The probabilistic roadmap method as described in this report requires a preparatory step where the roadmap is constructed for the area where the robot is to operate. Another approach is to construct a roadmap online for one specific planning problem. Whereas the roadmap construction normally takes quite some time this time could probably be reduced significantly by using a non-uniform distribution for randomly placing configurations mainly in the vicinity of the start- and goal- configuration and the area between them.

By creating the roadmap incrementally, some advantages are gained. The first that springs to mind is the ability to directly incorporate new data from the world, e.g. data gathered from the sensor inputs of the helicopter.

Another advantage is that additional constraints can easily be placed on the plans, such as requiring the helicopter to stay below a certain altitude or at all times be visible from a ground operator situated at a certain location. With the current approach these kinds of constraints would in many cases have to be built into the roadmap in advance which results in a less flexible operation of the helicopter.

5.1.4 Other approaches

A final remark on areas for further investigation is that the probabilistic roadmap method might not be the optimal method for path planning in this domain. As described in the literature the main advantage of the probabilistic roadmap method is its ability to handle high dimensional configuration spaces with narrow passages. The velocity-part of the helicopter-state does not stand for true dimensions of the configuration space but is instead imposing non-holonomic constraints. This leaves us with just the three dimensions of space. In the open type of environment where the helicopter will generally operate, most obstacles can be seen as just elevation of the ground, which would imply that the path-planning problem would rather be something that could be called a $2^1/_2$-dimensional problem. This might be possible to utilize.

5.2 Conclusion

The probabilistic roadmap approach, in its basic form, does produce satisfactory plans for most queries. However, as a result of the restricted connection-possibilities with the current controller, a path is seldom found through more narrow passages with roadmaps of reasonable size. The two stage algorithm for non-holonomic robots, solves this problem to a great extent, because of the increased simplicity in connecting nodes together, which gives a much more dense roadmap without increasing the number of nodes and connections.

The most problematic part with the path planner developed during this thesis work is the discrepancy between the planned paths and the actual executed paths. In order to be able use this path planner with the real physical helicopter this problem should be taken care of in some manner, and one possible solution is proposed in the end of chapter 2.

The third main component of the path planner, the collision checker, solves the problem successfully and only minor modification was proposed.

Appendix A

The Cubic Equation

A.1 Solution

The solutions to the cubic equation is calculated following the solution from [2]. The solution assumes that the equation is on the form

$$b_0 + b_1 t + b_2 t^2 + t^3 = 0 \tag{A.1}$$

which can be achieved by some minor rearrangements of equation 3.5:

$$\frac{a_0 - d}{a_3} + \frac{a_1}{a_3} t + \frac{a_2}{a_3} t^2 + t^3 = 0 \tag{A.2}$$

Identifying the coefficients of the polynomials in equation A.1 and A.2 we get

$$b_0 = \frac{a_0 - d}{a_3} \tag{A.3}$$

$$b_1 = \frac{a_1}{a_3} \tag{A.4}$$

$$b_2 = \frac{a_2}{a_3} \tag{A.5}$$

The next step is to calculate some intermediate variables in which the solutions will be described:

$$q = \frac{3b_1 - b_2^2}{9} \tag{A.6}$$

$$r = \frac{9b_2b_1 - 27b_0 - 2b_2^3}{54} \tag{A.7}$$

$$D = q^3 + r^2 \tag{A.8}$$

$$\tag{A.9}$$

Depending on the discriminator, D, we have three cases:

i) If $D > 0$, there is one real solution

$$t = -\frac{1}{3}a_2 + \sqrt[3]{r + \sqrt{D}} + \sqrt[3]{r - \sqrt{D}} \tag{A.10}$$

ii) If $D < 0$, there are three real solutions

$$t_1 = 2\sqrt{-q}\cos\left(\frac{\theta}{3} - \frac{1}{3}a_2\right) \tag{A.11}$$

$$t_2 = 2\sqrt{-q}\cos\left(\frac{\theta + 2\pi}{3} - \frac{1}{3}a_2\right) \tag{A.12}$$

$$t_2 = 2\sqrt{-q}\cos\left(\frac{\theta + 4\pi}{3} - \frac{1}{3}a_2\right) \tag{A.13}$$

$$\tag{A.14}$$

where $\theta = \arccos\left(\frac{r}{\sqrt{-q^3}}\right)$

iii) If $D = 0$, there are two sub cases

a) if $3a_1 - a_2^2 = 0$, there is one solution:

$$t = -\frac{a_2}{3} \tag{A.15}$$

b) otherwise there are two:

$$t_1 = \frac{a_1 a_2 - 9a_0}{2(3a_1 - a_2^2)} \tag{A.16}$$

$$t_2 = \frac{a_2^3 - 4a_1 a_2 + 9a_0}{3a_1 - a_2^2} \tag{A.17}$$

A.2 Computational cost

Calculation	Equation	Mul	Add
Conversion to standard form:	A.3–A.5	3	1
$b_0 + b_1 t + b_2 t^2 + t^3 = 0$			
Calculation of q	A.6	3	1
Calculation of r	A.7	7	2
Calculation of D	A.8	3	1
Sum:		16	5

Depending on which of the cases we have we will get one of the following additional costs:

i) $D > 0$

Calculation	Equation	Mul	Add	Func
Calculation of t	A.10	1	4	3
Sum:		1	4	3

ii) $D < 0$

Calculation	Equation	Mul	Add	Func
Calculation of $2\sqrt{-q}$	A.11–A.13	1	0	1
Calculation of t_1	A.11	3	1	1
Calculation of t_2	A.12	3	2	1
Calculation of t_3	A.13	3	2	1
Sum:		10	5	4

iii) For the two sub cases when $D = 0$ we have

a)

Calculation	Equation	Mul	Add	Func
Calculation of t	A.15	1	0	0
Sum:		1	0	0

	Calculation	Equation	Mul	Add	Func
	Calculation of $3a_1 - a_2^2$	A.16–A.17	2	1	0
	Calculation of a_1a_2	A.16–A.17	1	0	0
b)	Calculation of $9a_0$	A.16–A.17	1	0	0
	Calculation of t_1	A.16	2	1	0
	Calculation of t_2	A.17	3	3	0
	Sum:		9	5	0

Bringing these parts together the total of the operations in the different cases sums up to the following:

Calculation	Equation	Mul	Add	Func
$D > 0$		17	9	3
$D < 0$		26	10	4
$D = 0$ a)		17	5	0
$D = 0$ b)		25	10	0

Bibliography

[1] B. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, , and P. Raghavan. A random sampling scheme for robot path planning. In G. Giralt and G. Hirzinger, editors, *Proc. of the 7th International Symposium on Robotics Research*, pages 249–264. Springer, New York, 1996.

[2] Cubic equation, real variable — complete solution. `http://www.danbbs.dk/~kald1/cubic/Cubic.htm`, accessed 2001-07-01. This URL is unfortunately no longer valid 2002-01-15.

[3] 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.

[4] Patrick Doherty, Patrik Hasslum, Torsten Merz, Erik Skarman, Gianpaolo Conte, Simone Duranti, Fredrik Heintz, Per Olof Pettersson, Tommy Persson, and Björn Wingman. A distributed architecture for intelligent unmanned aerial vehicle experimentation. submitted to IAAI03, 2003.

[5] 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. Car-

olina, Chapel Hill, 1996. URL: `ftp://ftp.cs.unc.edu/pub/users/manocha/PAPERS/COLLISION/obb.ps.gz` accessed 11/05/2001.

[6] 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.

[7] T Siméon J.-P. Laumond. Notes on visibility roadmaps and path planning. Workshop on Algorithmic Foundations of Robotics, 2000. available on `http://www.laas.fr/~nic/publi.html`.

[8] 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. available on `http://www.cs.rice.edu/CS/Robotics/publications.html`.

[9] Lydia Kavraki and Jean-Claude Latombe. Randomized preprocessing of configuration space for path planning: Articulated robots. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS)*, pages 1764–1771, 1994. available on `http://www.cs.rice.edu/CS/Robotics/publications.html`.

[10] Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transaction on Robotics and Automation*, 12(4):566–580, 1996. available on `http://www.cs.rice.edu/CS/Robotics/publications.html`.

[11] Mark H. Overmars and Petr Švestka. A paradigm for probabilistic path planning. Technical report, Department of Computer Science, Utrecht University, 1996. `http://www.cs.ruu.nl/~markov/research.html`.

[12] David F. Rogers and J. Alan Adams. *Mathematical Elements for Computer Graphics*. McGraw-Hill Publishing Company, 2nd edition, 1990.

[13] Stuart Russel and Peter Norvig. *Artificial Intelligence, a Modern Approach*. Prentice Hall, Inc., 1995.

[14] Sepanta Sekhavat and Mark. H. Overmars Petr Švestka, J.-P. Laumond. Multi-level path planning for nonholonomic robots using semiholonomic subsystems, 1996. available on `http://www.cs.uu.nl/people/markov/research.html`.

[15] Erik Skarman. Personal communication.

[16] Erik Skarman. Helicopter control design. Internal technical report, June 2001.

[17] C.Nissoux T.Siméon, J.P.Laumond. Visibility based probabilistic roadmaps. IEEE Int. Conf. on Int. Robots and Systems, 1999. available on `http://www.laas.fr/~nic/publi.html`.

[18] C.Nissoux T.Siméon, J.P.Laumond. Visibility-based probabilistic roadmaps for motion planning. *Journal of Advanced Robotics*, 14(6):477–494, 2000. available on `http://www.laas.fr/~nic/publi.html`.

	Avdelning, Institution Division, Department AIICS, Dept. of Computer and Information Science 581 83 Linköping	**Datum** Date January 28, 2003

Språk Language ☐ Svenska/Swedish ☒ Engelska/English ☐ ______________	**Rapporttyp** Report category ☐ Licentiatavhandling ☒ Examensarbete ☐ C-uppsats ☐ D-uppsats ☐ Övrig rapport ☐ ______________	**ISBN** — **ISRN** — **Serietitel och serienummer** Title of series, numbering **ISSN** — LITH-IDA-EX-02-56
URL för elektronisk version		

Titel Helikopterruttplanering med stokastiska ruttkartor

Title Helicopter Path Planning using Probabilistic Roadmaps

Författare Author Per Olof Pettersson

Sammanfattning
Abstract

Probabilistic roadmap algorithms work in two stages. In the first, by randomly choosing collision free configurations of a robot and then attempting to connect these, a graph is created which represents the free space of the robot's environment. This graph can then be used to answer path planning queries during runtime.

In this report I describe an adaptation of this algorithm to the domain of an autonomously flying helicopter. I also present two extensions to the algorithm: visibility based probabilistic roadmaps and a method of initially relaxing the non-holonomic constraints of the robot.

The probabilistic roadmap algorithm requires a local path planner that is able to plan paths that take the specific robot, in this case a helicopter, from one state to a nearby one. In this report I present two variants of such a local planner that generates a flight path described as a parameterized cubic curve.

A third component necessary for path planning is a method for checking if a certain path is collision free. For this purpose a divide and conquer algorithm was used that bounds the whole model in one box and then recursively splits the box into smaller boxes. A path can then be checked for collision by solving for the intersections of the curve and the planes bounding these boxes, recursing down the tree-structure.

I show that this type of algorithm can successfully give answers to path planning queries in this domain, especially with the initial relaxation of non-holonomic constraints, which makes it possible to find paths even through very narrow passages.

Nyckelord Keywords Path planning, Probabilistic Roadmaps, Collision checking, OBBTree, Helicopters