Master Thesis

Path Planning for Fixed-Wing Unmanned Aerial Vehicles

Author(s):
Schneider, Daniel

Publication Date:
2016

Permanent Link:
https://doi.org/10.3929/ethz-a-010646508

Rights / License:
In Copyright - Non-Commercial Use Permitted
Master Thesis

Path Planning for Fixed-Wing Unmanned Aerial Vehicles

Autumn Term 2016

Supervised by:
Philipp Oettershagen
Enric Galceran Yebenes

Author:
Daniel Schneider
Declaration of Originality

I hereby declare that the written work I have submitted entitled

Path Planning for Fixed-Wing Unmanned Aerial Vehicles

is original work which I alone have authored and which is written in my own words.¹

Author(s)

Daniel Schneider

Student supervisor(s)

Philipp Oettershagen
Enric Galceran Yebenes

Supervising lecturer

Roland Siegwart

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Uttigen, 02 May 2016

Place and date	Signature

¹Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iv</td>
</tr>
<tr>
<td>Symbols, Notation, Terms And Definitions</td>
<td>vi</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Goal of this thesis</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Contributions</td>
<td>2</td>
</tr>
<tr>
<td><strong>2 Background</strong></td>
<td>4</td>
</tr>
<tr>
<td>2.1 Terms and definitions</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Motion planning</td>
<td>6</td>
</tr>
<tr>
<td>2.2.1 Problem formulation</td>
<td>6</td>
</tr>
<tr>
<td>2.2.2 Sampling-based motion planning</td>
<td>7</td>
</tr>
<tr>
<td>2.2.3 Geometric planning</td>
<td>9</td>
</tr>
<tr>
<td>2.2.4 Control-based planning</td>
<td>9</td>
</tr>
<tr>
<td>2.2.5 Informed sampling</td>
<td>13</td>
</tr>
<tr>
<td>2.2.6 Sampling-based motion planning algorithms</td>
<td>14</td>
</tr>
<tr>
<td>2.3 Optimality</td>
<td>19</td>
</tr>
<tr>
<td>2.4 Collision checking</td>
<td>21</td>
</tr>
<tr>
<td>2.4.1 Flexible Collision Library</td>
<td>21</td>
</tr>
<tr>
<td>2.4.2 Height checking</td>
<td>21</td>
</tr>
<tr>
<td>2.5 System dynamics</td>
<td>22</td>
</tr>
<tr>
<td>2.5.1 Dubins car</td>
<td>22</td>
</tr>
<tr>
<td>2.5.2 Dubins airplane</td>
<td>23</td>
</tr>
<tr>
<td><strong>3 Path planning method</strong></td>
<td>28</td>
</tr>
<tr>
<td>3.1 Software structure</td>
<td>29</td>
</tr>
<tr>
<td>3.2 Collision checking</td>
<td>29</td>
</tr>
<tr>
<td>3.3 Airplane motion model</td>
<td>30</td>
</tr>
<tr>
<td>3.3.1 Non-optimal Dubins airplane paths</td>
<td>31</td>
</tr>
<tr>
<td>3.3.2 Dubins set classification</td>
<td>32</td>
</tr>
<tr>
<td>3.3.3 Dubins airplane state space for OMPL</td>
<td>32</td>
</tr>
<tr>
<td>3.4 Path planning</td>
<td>33</td>
</tr>
<tr>
<td>3.4.1 Control-based planning</td>
<td>35</td>
</tr>
<tr>
<td>3.4.2 Informed sampling</td>
<td>36</td>
</tr>
<tr>
<td>3.4.3 Dubins airplane adapted sampling</td>
<td>36</td>
</tr>
<tr>
<td>3.4.4 Combining the use of straight lines and non-optimal Dubins airplane paths</td>
<td>37</td>
</tr>
<tr>
<td>3.4.5 Forward propagation</td>
<td>38</td>
</tr>
<tr>
<td>3.4.6 Optimal fast marching tree using non-optimal Dubins airplane paths (D-FMT*)</td>
<td>39</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>3.5</td>
<td>Manual code optimization</td>
</tr>
<tr>
<td>4</td>
<td>Experiments and results</td>
</tr>
<tr>
<td>4.1</td>
<td>Parameter tuning for planners</td>
</tr>
<tr>
<td>4.2</td>
<td>Preliminary results</td>
</tr>
<tr>
<td>4.2.1</td>
<td>FCL and height checking</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Determine the time complexity of FMT*</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Non-optimal vs. optimal Dubins airplane paths</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Manual code optimizations</td>
</tr>
<tr>
<td>4.3</td>
<td>Final results</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Block experiment</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Gap experiment</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Random map experiment</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Slalom experiment</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Mountain experiment</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Kandervalley experiment</td>
</tr>
<tr>
<td>4.4</td>
<td>Summary and analysis of the experiments</td>
</tr>
<tr>
<td>5</td>
<td>Conclusion and future work</td>
</tr>
<tr>
<td>5.1</td>
<td>Conclusion</td>
</tr>
<tr>
<td>5.2</td>
<td>Future work</td>
</tr>
<tr>
<td>A</td>
<td>Additional material</td>
</tr>
<tr>
<td>A.1</td>
<td>Proof: Eq. [3.6] is a pseudometric</td>
</tr>
<tr>
<td>A.2</td>
<td>Proof: The length of non-optimal Dubins airplane paths is a premetric</td>
</tr>
<tr>
<td>B</td>
<td>More results</td>
</tr>
<tr>
<td>B.1</td>
<td>Block experiment</td>
</tr>
<tr>
<td>B.2</td>
<td>Gap experiment</td>
</tr>
<tr>
<td>B.3</td>
<td>Random map experiment</td>
</tr>
<tr>
<td>B.4</td>
<td>Slalom experiment</td>
</tr>
<tr>
<td>B.5</td>
<td>Mountain experiment</td>
</tr>
<tr>
<td>B.6</td>
<td>Kandervalley experiment</td>
</tr>
</tbody>
</table>
Abstract

This thesis presents a first implementation of a real-time capable onboard path planning framework computing shortest paths for fixed-wing aerial vehicles for many start-goal configurations. The framework provides kinodynamic planning based on the Dubins airplane motion model.

The framework is implemented as a ROS node what makes it executable on common companion-computers installed on fixed-wing aerial vehicles. We evaluated 8 different sampling-based motion planning algorithms in theoretical and real experimental setups. Two new ideas to reduce the time for finding shortest paths for the Dubins airplane are explained and compared against existing planning methods. On the one hand, the informed subset of the Dubins airplane for minimum path length is approximated as a subset of the informed subset for a system without differential constraints. On the other hand, in order to find initial paths as fast as possible, the initial path is planned using straight-line connections between samples instead of dynamically feasible paths. A modification to the optimal Fast Marching Tree algorithm (FMT*) is presented. The modification allows to predict the amount of samples workable in the time available for planning and thereby makes FMT* usable for real-time and onboard path planning.

Experiments have shown that planning initial paths with straight lines and approximating the informed subset for the Dubins airplane for minimum path length allows finding initial paths in less than 1 second and expedites the convergence to the optimal path such that after 2 seconds reasonably short paths are found. Furthermore, planners iteratively drawing one sample at a time, e.g. optimal Rapidly-exploring Random Tree (RRT*), perform better than planners drawing batches of samples, e.g. Batch Informed Tree (BIT*), if planning in wide open spaces. For cluttered, narrow and twisted maps, it is the other way round. Additional experiments have shown that computing non-optimal paths for start-goal configurations occurring rarely in real application scenarios, speeds up the computation of the paths up to 40 times. Further experiments have shown that using a light collision checking algorithm reduces the time spent for collision checking more than 10 times.
I would first like to thank my thesis advisors, Philipp Oettershagen and Enric Galceran Yebenes of the Autonomous Systems Lab at ETH in Zurich, who steered me in the right direction and supported my work with a lot of helpful answers and bright ideas.
I also owe much to Prof. Dr. Roland Siegwart, the Autonomous Systems Lab and all members thereof, who made this thesis possible and in some way promoted it.
Finally, I must express my profound gratitude to my parents, siblings, cousins and friends for providing me with permanent support and continuous encouragement throughout this thesis.
Symbols, Notation, Terms And Definitions

Symbols

\( \phi, \psi, \theta \)  
Roll, pitch and yaw angle

\( \mathcal{C} \)  
Configuration space

\( q \)  
Configuration

\( q_I \)  
Start/ initial configuration

\( q_G \)  
Goal configuration

\( X \)  
State space

\( x \)  
State

\( x_I \)  
Start/ initial state

\( x_G \)  
Goal state

\( x \)  
x-component of a configuration \( q \) or state \( x \)

\( y \)  
y-component of a configuration \( q \) or state \( x \)

\( z \)  
z-component of a configuration \( q \) or state \( x \)

\( \Theta \)  
Yaw-component of a configuration \( q \) or state \( x \)

\( c_{\min} \)  
Minimum possible cost of a path connecting start and goal in obstacle-free space

\( c_{\text{best}} \)  
Cost of current best path

\( r_{\min} \)  
Minimum turning radius of the Dubins airplane

\( \gamma_{\max} \)  
Maximum climb and sink angle of the Dubins airplane

Acronyms and Abbreviations

ETH  
Eidgenössische Technische Hochschule

ASL  
Autonomous Systems Lab

UAV  
Unmanned aerial vehicle

RRT  
Rapidly-Exploring Random Tree

FMT  
Fast-Marching Tree

BIT  
Batch Informed Tree

KPIECE  
Kinodynamic Motion Planning by Interior-Exterior Cell Exploration

BVP  
Boundary value problem

RRT  
Rapidly-exploring Random Tree

RRT*  
Optimal Rapidly-exploring Random Tree

FMT*  
Optimal Fast Marching Tree

BIT*  
Batch Informed Tree
Chapter 1

Introduction

Autonomous flying requires knowing a path leading the way. The path not only has to be collision-free but also flyable by the aerial vehicle and it is desirable that the path is optimal in some sense. Fixed-wing aerial vehicles have the properties of not being able to stop on the spot, fly sharp curves and climb or sink without covering some distance horizontally. Due to these reasons, fixed-wing aerial vehicles most often are not suitable for indoor use. Hence, the vast amount of applications of fixed-wing unmanned aerial vehicles are outdoors in wide open spaces. This may be above the sea (human and commodity transport), above agricultural areas (field inspections), in mountainous regions (mountain rescues) or above cities (emergency aid or parcel delivery). Possible obstacles in these areas are large trees, mountains and skyscrapers.

In this thesis a path planning framework was implemented which yields collision-free and flyable paths for fixed-wing unmanned aerial vehicles. In many cases, the paths are optimal in the sense of minimum path length.

1.1 Motivation

This thesis was accomplished within the fixed-wing group of the Autonomous Systems Lab at ETH in Zurich.

Common small-scale unmanned aerial vehicles (UAVs) require the manual assignment of waypoints via a ground station operator (e.g. QGroundControl [1]). A procedure to produce paths is, to connect the waypoints with straight lines. Each waypoint has an acceptance radius. When the fixed-wing aerial vehicle is close enough to the point, it starts following the next straight line segment. In general, the paths manually generated are neither dynamically feasible for the airplane nor optimal. In order to cope with the dynamic feasibility, the acceptance radius was introduced. However, this makes it more difficult to ensure collision-free paths. The optimality of the path is dependent on the sophisticatedness of the person choosing waypoints. Furthermore, choosing optimal waypoints is even more tricky when introducing the acceptance radius.

For the sake of autonomy, dynamic feasibility and optimality, the procedure to plan paths should be automated. This is done by path planning algorithms.
1.2 Goal of this thesis

The goal of this thesis was to develop a first implementation of a path planning framework computing collision-free paths minimizing path length, flyable by fixed-wing aerial vehicles. The framework has to be real-time capable and compatible with common companion-computers used on fixed-wing UAVs (sec. 3). The thesis includes the evaluation of different path planning algorithms for the scenario of minimizing path length but also for the scenario with an arbitrary optimization objective. The planners have to satisfy real-time requirements, such as finding an initial path in less than 1 second and providing an improved short path after about 2 seconds.

An example of a collision-free, dynamically feasible path found after two seconds of planning is shown in fig. 1.1.

1.3 Contributions

Our contributions to the framework are as follows:

• We implemented a path planning framework for fixed-wing unmanned aerial vehicles in C++, using ROS and the open motion planning library (OMPL) (sec. 3.1).

• We extended the Dubins car state space provided by OMPL to three dimensions and adapted it for the planning with the Dubins airplane (sec. 3.3.3).

• We applied the concept of approximating the informed subset for a specific dynamic system and a specific cost function to the Dubins airplane (sec. 3.4.3).

• In order to find initial paths as fast as possible, we neglected the optimization objective during the planning of an initial path (sec. 3.4.4). Globally asymptotically optimal behavior is not impaired due to subsequent planning with considering the optimization objective.

• We modified Fast Marching Tree (FMT*) to make the planner real-time capable (sec. 3.4.6).

• We evaluated the different planners with various experiments including theoretical and real maps (sec. 4).
Figure 1.1: A collision-free, dynamically feasible path for a fixed-wing aerial vehicle around a mountain in Valais, Switzerland, after 2 seconds of planning. The position and orientation of the start and goal are shown as green and red, respectively. An initial path was found in less than 0.1 seconds. After 2 seconds, the length of the improved planned path (red line) is 120% of the length of the shortest dynamically feasible path connecting start and goal.
Chapter 2

Background

The background section covers the main theoretical background of the scientific contributions on which this thesis is founded.

2.1 Terms and definitions

In this section, some specific terms used in this thesis are briefly elucidated.

- 2.5D map: Also called height map or digital elevation map (DEM). A 2.5D map is a discrete representation of a surface. It contains the height of a surface at discrete points. Every point defines a square cell with a side length of the distance to its nearest point. The whole cell is assumed to have the height of the midpoint of the cell.

- Asymptotic optimality: A sampling-based path planning algorithm is said to be asymptotically optimal if the cost of the solution it returns converges almost everywhere to the optimal cost as the number of samples approaches infinity \[2\].

- Configuration space $C$: The parameters defining a configuration of a system are called generalized coordinates. The vector space defined by the generalized coordinates is called the configuration space of a system. The dimension of the configuration space is the same as the number of degrees of freedom of the robot (velocity and acceleration are usually not part of the configuration space). Examples of configuration spaces are
  - Particle moving in the three-dimensional euclidean space: $C = \mathbb{R}^3$
  - A rigid body: $C = \mathbb{R}^3 \times \text{SO}(3)$

- $C_{\text{obs}}$: The obstacle occupied configuration space $C_{\text{obs}}$ is a subspace of $C$. It contains configurations that represent a collision of the robot with obstacles in the world $\mathcal{W}$ or with itself (e.g. robots with links and joints).

- $C_{\text{free}}$: $C \setminus C_{\text{obs}}$

- Differential constraint: A differential constraint is a constraint which is not integrable into a positional constraint ($f(\mathbf{q}, \dot{\mathbf{q}}, t) = 0$).

- Geometric constraint: A constraint on the position of the robot.

- Kinematic constraint: A constraint on the movement of components of a mechanical system. A kinematic constraint can be a holonomic or a non-holonomic constraint.
• Kinodynamic motion planning: The kinodynamic motion planning problem is to synthesize a robot motion subject to simultaneous kinematic constraints such as avoiding obstacles, and dynamics constraints, such as modulus bounds on velocity, acceleration and force. Donald et al. [3] introduce the term kinodynamic motion planning in more details.

• Long path case: The long path case is defined for the Dubins car. A start and goal state for the Dubins car are said to be a long path case, if the shortest path connecting them (the Dubins car path) is one of the following Dubins car path types: RSR, RSL, LSR, LSL. This is the case if the start and goal are far enough apart from each other. For an exact definition, consider Shkel and Lumelsky’s work [4]. In this thesis, long path case is also used to refer to non-optimal Dubins airplane paths for which the underlying path is one of RSR, RSL, LSR, LSL when computing the paths with algs. 9, 10, 11.

• Metric / distance function: A metric or distance function is a function that defines a distance between each pair of elements of a set. The distance function has to fulfill four conditions: non-negativity, identity of indiscernibles, symmetry, triangle inequality. Willard introduces metrics formally [5].

• Motion: A motion consists of a configuration and a trajectory. Each motion has a cost assigned. A motion can have a parent (only one parent if the search graph is a tree) and children. A motion corresponds to an edge in the search graphs of the motion planning algorithms.

• (Non-)holonomic constraint: A holonomic constraint is a constraint which is integrable into a positional constraint (f(q, t) = 0). A non-holonomic constraint is the same as a differential constraint.

• Obstacle region O: The part of the world W which is occupied with obstacles [6].

• Premetric: A metric for which the triangle inequality and the symmetry condition (see metric / distance function) is not fulfilled, e.g. non-optimal Dubins airplane paths.

• Probabilistic completeness: A sampling-based path planning algorithm is said to be probabilistically complete if the probability of finding a solution, if one exists, converges to one as the number of samples approaches infinity [2].

• Quasimetric: A metric for which the symmetry condition (see metric / distance function) is not fulfilled, e.g. the length of Dubins car and Dubins airplane paths.

• State space X: The set of possible states of a system. For a pure algebraic system, the state space is the same as the configuration space. For dynamic physical system, the state space typically includes position and momentum variables (e.g. velocity).

• Steering function: A function computing the control input to a dynamic system and the corresponding state trajectory from a start state to a goal state.

• World W: The world, usually $\mathbb{R}^2$ or $\mathbb{R}^3$.

\footnote{The extension of the term “long path case” for the Dubins airplane is more complicated when the paths are optimal also for the intermediate case.}
2.2 Motion planning

This section presents the basic problem formulation of motion planning. Furthermore, an overview of different approaches to solve the challenge of motion planning, relevant for this thesis, is given.

2.2.1 Problem formulation

The original problem formulation for the motion planning task, called the piano mover’s problem, did not consider system dynamics. Intuitively, it can be formulated as follows: Find a path from $q_I$ to $q_G$ in $C_{free}$, where $q_I$ and $q_G$ are the start and goal configuration of the robot, both in $C_{free}$ (fig. 2.1). Lavalle [6] introduces the piano mover’s problem formally. The piano mover’s problem does not impose any constraints on the transition between two configurations. Fig. 2.1 shows an example path in the obstacle-free configuration space satisfying the requirements of the piano mover’s problem, however, for a dynamic system like a fixed-wing aerial vehicle, the resulting path is not flyable because of a violation of the minimum turn radius constraint of fixed-wing aerial vehicles.

Figure 2.1: The basic motion planning problem. Find a path from the initial configuration $q_I$ to the goal configuration $q_G$ in the obstacle-free configuration space $C_{free}$. The obstacle region $C_{obs}$ is colored in red.

Therefore, for dynamic systems, the state space $X$ and the control input space $U$ is introduced. In these spaces, the dynamics of the system $\dot{x} = f(x, u)$ are described. This thesis has the goal to plan paths for fixed-wing aerial vehicles which are dynamic systems. Problem formulation [1] formally defines the task of motion planning under differential constraints, used for this thesis.

**Problem Formulation 1** Motion Planning Under Differential Constraints

1. A world $W$, a robot $A$ (or $A_1, A_2, \ldots, A_m$ for a linkage), an obstacle region $O$, and a configuration space $C$, which are defined the same as for the piano mover’s problem [6].

2. An unbounded time interval $T = [0, \infty)$.

3. A smooth manifold $X$, called the state space, which may be $X = C$ or it may be a phase space derived from $C$ if dynamics is considered. Let $bm : X \rightarrow C$
denote a function that returns the configuration \( q \in C \) associated with \( x \in X \). Hence, \( q = \kappa(x) \).

4. An obstacle region \( X_{\text{obs}} \) is defined for the state space. If \( X = C \), then \( X_{\text{obs}} = C_{\text{obs}} \). The notation \( X_{\text{free}} = X \setminus X_{\text{obs}} \) indicates the states that avoid collision and satisfy any additional global constraints.

5. For each state \( x \in X \), a bounded action space \( U(x) \subseteq \mathbb{R}^m \cup \{x_T\} \), which includes a termination action \( u_T \) and \( m \) is some fixed integer called the number of action variables. Let \( U \) denote the union of \( U(x) \) over all \( x \in X \).

6. A system is specified using a state transition equation \( \dot{x} = f(x, u) \), defined for every \( x \in X \) and \( u \in U(x) \). This could arise from any differential model. If the termination action is applied, it is assumed that \( f(x, u_T) = 0 \) (and no cost accumulates, if a cost functional is used).

7. A state \( x_I \in X_{\text{free}} \) is designated as the initial state.

8. A set \( X_G \subset X_{\text{free}} \) is designated as the goal region.

9. A complete algorithm must compute an action trajectory \( u : T \to U \), for which the state trajectory \( \bar{x} \), resulting from def. 2, satisfies: 1) \( x(0) = x_I \), and 2) there exists some \( t > 0 \) for which \( u(t) = u_T \) and \( x(t) \in X_G \).

If an optimization objective is important, a cost functional \( L \) of the form shown in eq. 2.1 can be used.

\[
L(\bar{x}_{t_F}, \bar{u}_{t_F}) = \int_0^{t_F} l(x(t), u(t)) \, dt + l_F(x(t_F))
\]

where

- \( \bar{x}_{t_F} \) is the state trajectory
- \( \bar{u}_{t_F} \) is the action trajectory
- \( t_F \) is the time at which the termination action is applied
- \( l(x, u) \) is the cost of a specific state \( x \) and input \( u \)
- \( l_F(x(t_F)) \) is the terminal cost

The challenge of motion planning, formulated in problem formulation [1] can be interpreted as a boundary value problem (BVP). The start state and goal region define the boundary conditions. The objective is to find a trajectory in the obstacle-free configuration space \( C_{\text{free}} \), satisfying the differential constraints. Using eq. 2.1, the BVP is extended to find optimal paths.

In order to solve the challenge formulated in the problem formulation [1], the world, the robot, the obstacle region, the configuration space, the state space, the action space, the state transition equation, the initial state and the goal region have to be specified. In the following sections, a review on the literature describing tools (collision checking, system dynamics and motion planning algorithms) to solve the challenge formulated in the problem formulation [1] is given. Together, these tools can be used to solve motion planning tasks. Only tools which are relevant for this thesis (sec. 1.2) are presented.

### 2.2.2 Sampling-based motion planning

The main idea of sampling-based motion planners is the following:

2This idea makes sampling-based motion planning different from deterministic motion planning. Hwang and Ahuja [7] and Latombe [8] describe deterministic motion planning in more details.
• Do not explicitly construct the part of configuration space \( C \) which yields collision \( C_{\text{obs}} \) and deterministically search for a path through the obstacle-free configuration space \( C_{\text{free}} \).

• Instead, draw samples from the configuration space \( C \) and check each sampled state for collision (sec. 2.4).

Hence, sampling-based motion planning algorithms are independent of the geometric models of \( C_{\text{obs}} \). Alg. 1 shows the high-level procedure of many sampling-based motion planning algorithms.

Algorithm 1 General procedure of sampling-based motion planning. The procedure is described in more details in [6]. If an optimization criterion is important, the cost functional (eq. 2.1) has to be considered in step 4, 5 and 6, in order to guarantee optimal connections between pairs of states and optimal construction of the search graph.

1: procedure SAMPLING_BASED_PLANNING\((x_I, x_G)\)
2: Initialization: Add the start state \( x_I \in X_{\text{free}} \) and possibly other collision-free states (such as \( x_G \)) to the undirected search graph \( \mathcal{G}(V,E) \).
3: while Not reached goal region and no other termination condition satisfied do
4: Vertex Selection Method: Choose a vertex \( x_{\text{cur}} \in V \) for expansion (e.g. by randomly sampling a state \( x_r \in X_{\text{free}} \) and searching its nearest neighbor \( x_{\text{near}} \in V \) according to some cost functional (eq. 2.1), then \( x_{\text{cur}} = x_{\text{near}} \)).
5: Local Planning Method: Construct a state trajectory \( \tau \) (possibly this trajectory has to satisfy the differential constraints) from \( x_{\text{cur}} \) to some state \( x_{\text{new}} \in X_{\text{free}} \) (e.g. \( x_{\text{new}} = x_r \)). Check if \( x_{\text{cur}}, x_{\text{new}} \) and \( \tau \) are in \( C_{\text{free}} \). If they are not, reject \( \tau \) and \( x_{\text{new}} \) and go to step 3. (For dynamical feasibility, consider sec. 2.5 for collision-free trajectories, consider sec. 2.4.)
6: Insert an Edge in the Graph: Insert \( \tau \) into \( E \), as an edge from \( x_{\text{cur}} \) to \( x_{\text{new}} \). If \( x_{\text{new}} \) is not yet in \( V \), insert it into \( V \).
If necessary, perform rewirings in \( \mathcal{G} \) in order to assure that the trajectory from the start vertex \( x_I \) to each vertex in \( \mathcal{G} \) is optimal.
7: end while
8: Return current best path or failure
9: end procedure

There exist many variations to this procedure, especially in the vertex selection method and the local planning method, but also in the other steps of the algorithm 1 (sec. 2.2.6). Lavalle [6], Elbanhawi and Simic [9] and en Muestreo and Compendio [10] give good overviews of different sampling-based motion planning algorithms. Some ideas which are important for this thesis are listed below:

• Instead of sampling one sample at a time, some sampling-based motion planning algorithms draw batches of samples all at once. Instead of connecting only the nearest neighbors (where nearest neighbors are searched using a cost functional), the k nearest neighbors are connected with edges. The path minimizing a cost functional is searched using a graph search algorithm (sec. 2.2.6 and 2.2.6).

• Another idea is to sample control-inputs instead of sampling states (sec. 2.2.6).
• Some algorithms also consider the system dynamics in the local planning method (sec. 2.2.4) [11].

2.2.3 Geometric planning

In this section, only sampling-based planning is considered. Geometric motion planning solves the piano mover’s problem. Geometric motion planning algorithms account for geometric and kinematic constraints of a system. In alg. 1, edges between vertices during the local planning step are constructed such that the geometric and kinematic constraints are satisfied. Typically, the edges are straight lines in the world $W$.

2.2.4 Control-based planning

In this section, only sampling-based planning is considered. Control-based motion planning solves the challenge formulated in problem formulation 1, i.e., it accounts for the differential constraints of a system. Referring to alg. 1, edges between sampled states have to satisfy the differential constraints. The edges are constructed in the local planning step. The local planning method, which is exactly the same task as formulated in 1 locally, can be interpreted as a BVP [6]. The objective is to find a state trajectory $\tilde{x}$. Depending on the problem, $\tilde{x}$ has to be optimal. The trajectory has to satisfy the boundary conditions of starting at the state $x_{\text{cur}}$ and ending at the state $x_{\text{new}}$. Furthermore, the trajectory has to be in $C_{\text{free}}$ and satisfy the dynamics of the system (the differential constraints).

Two different procedures, both guaranteeing that the differential constraints are satisfied, are described in more details:

• Sample a state $x_{\text{new}} \in X_{\text{free}}$ and solve the BVP between $x_{\text{new}}$ and $x_{\text{cur}} \in V$, a state already in the graph. The BVP may have an optimization objective. Use the resulting state trajectory as an edge in the graph (sec. 2.2.4). This procedure is iterated to construct a graph of motions.

• Sample a control input $u \in U$ of a system and apply it for a user-defined or random amount of time. Use the resulting state trajectory as an edge in the graph (sec. 2.2.4). This procedure is iterated to construct a graph consisting of motions.

Alg. 2 shows the general procedure for control-based planning.

Forward propagation using ODEs

Alg. 3 shows a high-level procedure of motion planning with a forward propagation routine. The procedure is based on the mathematical description $\dot{x} = f(x, u)$ of a system. A graph of motions exploring the space is constructed by iteratively processing the following steps:

1. A state $x_{\text{cur}}$ from the existing graph of motions is randomly selected.

2. A control-input $u$ of the system is randomly sampled.

3. The system is forward propagated from $x_{\text{cur}}$ by applying the input $u$ for a user-defined or random amount of time.

4. The transitions is considered as an edge, and the end state after propagation as a new vertex.
Algorithm 2  General procedure of control-based motion planning. The procedure is described in more details in [6]. If an optimization criterion is important, the cost functional (eq. 2.1) has to be considered in step 4, 5 and 6, in order to guarantee optimal connections between pairs of states and optimal construction of the search graph.

1: procedure PLAN_CONTROL_BASED($x_I$, $x_G$, $dt_{max}$, ODE)
2: Initialization: Add $x_I$ and possibly other collision-free states to the vertex set $V$ of the search graph $G(V,E)$. The edge set $E$ is empty.
3: while Not reached goal region and no other termination condition satisfied do
4: Swath-point Selection Method: Choose a vertex $x_{cur} \in V$ for expansion (e.g. randomly choosing any $x_{cur} \in V$ or randomly sampling a state $x_r \in X_{free}$, searching its nearest neighbor $x_n \in V$ and choosing $x_{cur} = x_n$)
5: Local Planning Method: Generate a motion primitive $\tilde{u}_p : [0,t_F] \rightarrow X_{free}$ such that $u(0) = x_{cur}$ and $u(t_F) = x_{tf}$ for some $x_{tf} \in X_{free}$, which may or may not be a vertex in $G$. $\tilde{u}_p$ has to satisfy the differential constraints. Furthermore, check $\tilde{u}_p$ for collision. If it yields collision or does not satisfy the differential constraints, return to step 3, otherwise continue.
6: Insert an Edge in the Graph: Insert $\tilde{u}_p$ into $E$. If $x_{tf}$ is not yet in $V$, add it to $V$.
   If necessary, perform rewirings in $G$ in order to assure that the trajectory from the start vertex $x_I$ to each vertex in $G$ is optimal.
7: end while
8: Return current best path or failure
9: end procedure

Algorithm 3  A procedure for control-based motion planning using a forward propagation routine [12, 13]. Instead of sampling the control input, also a state $x_r$ can be randomly sampled as mentioned in alg. 1. A method steering the system from the state to expand from $x_{cur}$ towards the sampled state $x_r$ is then applied.

1: procedure PLAN_WITH_FORWARD_PROPAGATION($x_I$, $x_G$, $dt_{max}$, ODE)
2: Initialization: Add $x_I$ to the graph $G(V,E)$
3: while Not reached goal region or no other termination condition satisfied do
4: Swath-point Selection Method: Randomly select a vertex $v = x_{cur}$ from the vertex set $V$
5: Local Planning Method: Randomly sample an input $u$ to the system described by the ODE.
   Randomly sample a duration $dt = \text{unif}(0,dt_{max})$
   Apply $u$ starting at $v$ for a duration of $dt$
6: Insert an Edge in the Graph: Add the state trajectory to the edge set $E$ and the end state $x_{new}$ to $V$
7: end while
8: Return current best path or failure
9: end procedure
The main advantage of planning with a forward propagation routine is that motion planning can be done for any system describable with ordinary differential equations. The main disadvantages are that planning with a forward propagation routine does not give any guarantee on finding optimal paths and is computationally expensive.

**Solve BVP**

As mentioned in sec. 2.2.1, the task of motion planning can be viewed as a BVP. However, BVP solvers are not well-suited to handle obstacle regions \[6\]. The local planning task is exactly the same task as the problem formulation \[1\]. Thus, also the local planning problem can be viewed as a BVP. For the local planning task, an adequate assumption is that the world is locally obstacle-free (the obstacle region \( O \) is empty). Therefore, the BVP can be solved without considering obstacles. The objective is to find a trajectory \( \tilde{x} \) in the state space. The trajectory has to satisfy the boundary conditions at the start state, \( x_{\text{cur}} \), and at the end state, \( x_{\text{new}} \). Furthermore, the trajectory has to respect the kinematics and dynamics of the system, in other words, the kinematic and differential constraints. Due to the assumption of locally obstacle-free space, the BVP solver does not have to consider obstacles in the world \( W \). Instead, the result from the BVP is checked for collision and rejected in case it indeed yields a collision \[12\] (sec. 2.2.4). Furthermore, in the same way as described for task \[1\] also the BVPs in the local planning method can be solved with regard to an optimization criterion (eq. 2.1). In order to find optimal trajectories for a dynamic system from start to goal, it is a necessary condition for the local planning method in alg. \[1\] to be solved optimally \[14\]. For the local planning method, BVP solvers are better-suited since obstacles in \( W \) are not considered. However, also solving BVPs in obstacle-free spaces usually is costly, except if there exists a closed-form solution to the BVP.

**Closed-form solution to the BVP**

In this section it is assumed, that an optimal (with respect to eq. 2.1) closed-form solution to the BVP formulated in problem formulation \[1\] is known for arbitrary boundary conditions \( x_I \) and \( x_G \). In the local planning method (alg. 2), the trajectory between two states has to be computed. This can be done using the closed-form solution to the BVP for any two states. In other words, the edges of the graph are the result from the BVP between two vertices. Since the closed-form solution is known, the edges are rapidly computable. Note that each time an edge is computed, it has to be checked for collision, since we assumed that the world is locally obstacle-free (\( O \) is empty). Locally obstacle-free space is a reasonable assumption such that most trajectories resulting as a solution to the local BVP are collision-free.

For dynamic systems for which an optimal closed-form solution to the BVP is known for arbitrary boundary conditions, motion planning can be done efficiently. By constructing a graph as in alg. 1 and connecting states using the closed-form solution in the local planning method for optimal trajectories to the local BVP, the dynamics of the system are incorporated in the planning in an elegant way. The motion planning algorithm does not need to consider the kinematic and differential constraints of the system. Any geometric motion planning algorithm (sec. 2.2.3) can be used. Alg. 4 shows the procedure of planning when the solution to the BVP is known in closed-form.

Examples of dynamic systems for which the closed-form solution for optimal trajectories to the BVP formulated in \[1\] in the obstacle-free space is known:

- Dubins car: Dubins \[15\] showed that the closed-form solution for shortest length paths to the BVP formulated in \[1\] is known for arbitrary boundary conditions \( x_I, x_G \) (sec. 2.5.1).
Algorithm 4 General procedure of control-based motion planning when the solution to the BVP is known in closed-form. There exist different variations to this procedure.

1: **procedure** PLAN\_CONTROL\_BASED\_CLOSEDFORM\( (x_I, x_G) \)
2: **Initialization:** Add \( x_I \in X_{free} \) to the vertex set \( V \) of the *search graph* \( \mathcal{G}(V, E) \). The edge set is \( E \) is empty.
3: **while** Not reached goal region and no other termination condition satisfied 
   do
4:  **Swath-point Selection Method:** Randomly sample a state \( x_r \in X \). Check if \( \kappa(x_r) \in C_{free} \). If so, continue, otherwise, go to step 3.
   Search for its nearest neighbor \( x_{cur} \in V \). Nearest refers to the neighbor \( x_n \) for which the edge between \( x_n \) and \( x_r \) is optimal with respect to eq. [2.1]. Note that this requires solving the BVP for every state in \( V \). However, the solution to the BVP is known in closed-form, therefore these computations can be done fast.
5:  **Local Planning Method:** Using the closed-form solution to the BVP, compute the optimal trajectory from \( x_{cur} \) to \( x_r \). Check if the trajectory is collision-free (no state of the trajectory is in the obstacle region \( O \). If so, continue, otherwise reject \( x_r \) and the trajectory and return to step 3.
6:  **Insert an Edge in the Graph:** Insert the trajectory into \( E \). If \( x_r \) is not yet in \( V \), add it to \( V \). If necessary, perform rewirings in \( \mathcal{G} \) in order to assure that the trajectory from the start vertex \( x_I \) to each vertex in \( \mathcal{G} \) is optimal.
7: **end while**
8: **Return** current best path or failure
9: **end procedure**
• Dubins airplane: Chitsaz and Lavalle \cite{16} showed that the closed-form solution for shortest length paths to the BVP formulated in \cite{1} is known for many boundary conditions $x_I, x_G$ (sec. 2.5.2).

### 2.2.5 Informed sampling

Informed sampling is an idea which can be used in sampling-based motion planning algorithms (e.g. informed RRT* in sec. 2.2.6 or BIT* in sec. 2.2.6) as soon as a trajectory from start to goal is known. The word informed refers to the fact that the motion planning procedure knows when a trajectory from start to goal exists, and uses information about the cost of this trajectory.

The idea of informed sampling is that as soon as a trajectory between start and goal is found, it does not make sense to sample a state, for which the optimal trajectory from the start through the state to the goal has already a higher cost than the cost of the current best trajectory, assuming an obstacle free state space. Hence, the search can be focused on the part of the states space which only contains samples which potentially can improve the current best trajectory. This region is called informed subset (eq. 2.2).

$$X_c = \left\{ x \in X \mid c(x) < c_{\text{best}} \right\},$$

where $c(x)$ is the cost of an optimal path from start to goal constrained to go through $x$.

$c_{\text{best}}$ is the cost of the current best path (2.2)

The informed subset contains only states for which the optimal trajectory from the start through the state to the goal has a lower cost than the current best trajectory, if there were no obstacles in the state space. All optimal trajectories from the start through a state on the surface of the informed subset to the goal lead to a trajectory which has the same cost as the current best trajectory. In real applications, there usually are obstacles. Therefore, not every state in the informed subset yields a better trajectory. However, the search can be focused on a part of the whole state space. This yields much denser exploration of this part of the state space, thus it is more likely to find lower cost trajectories \cite{17}.

The informed subset can have different shapes.

• If planning for systems without differential constraints (these systems can pursue straight line paths with sharp corners) and if the optimization objective is path length, the informed subset has the form of an n-dimensional ellipsoid. An n-dimensional ellipsoid can directly and efficiently be sampled (fig. 2.2).

• If planning for an arbitrary system and the optimization objective is path length, the informed subset has complex shapes and in general cannot be sampled directly. However, the informed subset is contained in the informed subset which would result for a system without differential constraints. This is due to the fact that the path length between two states for systems without differential constraints (euclidean distance) is always shorter than the length of the paths for any other system between the same two states.

• For arbitrary systems and cost functionals, the informed subset has complex shapes and in general cannot be sampled directly.
Chapter 2. Background

2.2.2.6 Sampling-based motion planning algorithms

In the subsequent sections, some motion planning algorithms that are relevant for this thesis are described. The planners can be classified according to different criteria.

Geometric planners and control-based planners

- Geometric planners do not consider the dynamics of a system. These attempt to solve the piano mover’s problem. Geometric planners can be adapted for control-based planning when solving the BVP locally for each newly sampled state (sec. 2.2.4). In this case, the planners sample states instead of configurations (sec. 2.2.4). This is inefficient except in case a closed-form solution to the BVP is known.

- Control-based planners consider the dynamics of a system. They directly attempt to solve the task without the necessity to solve this BVP locally.

Optimizing and non-optimizing motion planning algorithms

- Some of the planners can handle arbitrary optimization objectives (eq. 2.1).

- Some of the planners work only for specific optimization objectives (e.g. path length as defined in eq. 2.4).

- Some of the planners cannot handle cost functions and do not optimize a criterion.

Motion planning algorithms sampling configurations / states one at a time and motion planning algorithms drawing batches of samples

- Planners consecutively drawing samples one at a time: RRT, RRT*, informed RRT*, KPIECE

- Planners using batches of samples: FMT*, BIT*

An overview of the different planners is given in tab. 2.1.

A note for all geometric planners presented in the following section: In case of using the geometric planners for control-based planning by solving local BVPs, configurations \( q \) and the configuration space \( C \) can be exchanged with states \( x \) and...
Table 2.1: Overview of the planners described in sec. 2.2.6

<table>
<thead>
<tr>
<th>Optimization Objective</th>
<th>Geometric planner</th>
<th>Control-based planner</th>
</tr>
</thead>
<tbody>
<tr>
<td>No optimization objective</td>
<td>RRT</td>
<td>KPIECE</td>
</tr>
<tr>
<td>Arbitrary optimization objective</td>
<td>RRT*, FMT*</td>
<td>-</td>
</tr>
<tr>
<td>Specific optimization objective</td>
<td>Informed RRT*, BIT*</td>
<td>-</td>
</tr>
</tbody>
</table>

the state space \( X \) respectively. This includes the special case of the configuration space being the same as the state space \( C = X \) (e.g. for the Dubins airplane).

Rapidly-exploring Random Tree (RRT)

- RRT is a probabilistically complete motion planning algorithm.
- RRT is not asymptotically optimal.
- RRT does not consider the dynamics of a system in the procedure itself. It can be adapted for planning under differential constraints by solving the BVP locally between any two states during the procedure (“SELECT \_INPUT(\( q_1, q_2 \))” in alg. 3).

The rough procedure of RRT is (fig. 2.3)

1. Sample a configuration \( q_r \) and search its nearest configuration \( q_n \) in the existing tree according to some predefined metric \( \rho \) (usually euclidean metric).
2. Connect \( q_n \) and \( q_r \) optimally according to \( \rho \) with a collision-free trajectory. Add a new configuration \( q_{\text{new}} \) to the tree (\( q_{\text{new}} \) is related to \( q_r \), usually a configuration in between \( q_n \) and \( q_r \)).

Figure 2.3: Example graph constructed by the RRT algorithm after 1, 2 and 3 seconds [18].

A high-level description of the RRT is given in alg. 5. Lavalle [18] describes the RRT algorithm in more details.

Optimal Rapidly-exploring Random Tree (RRT*)

The RRT* planning algorithm is an asymptotically optimal variant of RRT.

- RRT* is a probabilistically complete motion planning algorithm.
- RRT* can handle arbitrary cost functionals of the form eq. 2.1 as optimization objectives.
Algorithm 5 RRT Algorithm [18]: For a given initial configuration \( q_I \), a rapidly-exploring random tree \( T \) is constructed. The tree finally has \( K \) vertices.

1: procedure GENERATE_RRT(\( q_I \), \( K \), \( \Delta t \))
2: \( T.\text{init}(q_I) \)
3: for \( k = 1 \) to \( K \) do
4: \( q_r \leftarrow \text{RANDOM\_STATE()} \)
5: \( q_n \leftarrow \text{NEAREST\_NEIGHBOR}(q_r, T) \)
6: \( u \leftarrow \text{SELECT\_INPUT}(q_r, q_n) \)
7: \( q_{\text{new}} \leftarrow \text{NEW\_STATE}(q_n, u, \Delta t) \)
8: \( T.\text{add\_vertex}(q_{\text{new}}) \)
9: \( T.\text{add\_edge}(q_n, q_{\text{new}}, u) \)
10: end for
11: Return \( T \)
12: end procedure

- RRT* does not consider the dynamics of a system in the procedure itself. When solving the BVP locally between any two states during the procedure, RRT* is adapted for planning under differential constraints.

The main differences between RRT and RRT*, which make RRT* asymptotically optimal, are listed below:

- The RRT* algorithm has a rewiring step. This step makes sure that vertices in the graph which may be reached through a newly sampled vertex with lower cost, are reconnected via the new vertex while maintaining a tree structure of the graph. In this way, it is assured that the path from the start to any configuration in the tree is optimal in the tree.

- The radius for the near neighbors search in the rewiring step changes when the tree grows in order to prevent the number of near neighbor vertices to become arbitrarily large.


Informed optimal Rapidly-exploring Random Tree

The informed RRT* planning algorithm is an informed variant of RRT* for minimizing path length (sec. 2.2.5).

- Informed RRT* is a probabilistically complete motion planning algorithm.

- Informed RRT* is an asymptotically optimal motion planning algorithm. Informed RRT*, as presented by Gammell et al. [17], minimizes path length.

- Informed RRT* does not consider the dynamics of a system.

Informed RRT* works in the same way as the RRT* algorithm with the only difference that once a path is found, the space is sampled only in the region of configurations that potentially can improve the current best path (fig. 2.4). Eq. 2.3 expresses the informed subset for informed RRT* mathematically.

\[
C_c = \{ q \in C \mid ||q_I - q||_2 + ||q - q_G||_2 \leq c_{\text{best}} \} \tag{2.3}
\]

Gammell et al. [17] give a more detailed explanation of informed RRT*.
2.2. Motion planning

Figure 2.4: Informed sampling: Assuming the optimization objective is minimum path length, and knowing a path from start to goal, informed sampling focuses its sampling on an ellipsoidal part of the space \[17\].

**Fast Marching Tree (FMT*)**

- The FMT* is a probabilistically complete motion planning algorithm.
- FMT* is an asymptotically optimal motion planning algorithm. The original version of FMT* minimizes path length, however in general, FMT* can handle arbitrary cost functionals of the form eq. \[2.1\] as optimization objectives.
- FMT* does not consider the dynamics of a system in its original version. However, FMT* was extended to deal with driftless differential constraints \[20\] and with differential constraints even with drift but only for linear affine dynamics \[21\].

The rough procedure of FMT* is (fig. 2.5)

1. Sample a user-defined amount of configurations all at once. Additionally, start and goal configurations are added to the batch of samples.

2. Iteratively construct a graph starting at the start configuration. Two samples are considered as neighbors and are connected if the cost of the path connecting the two samples is lower than some connection radius. Doing so, concurrently, graph construction and dynamic-programming-style graph search is performed.

A high-level description of the FMT* is given in alg. \[6\]. Janson et al. \[19\] explain FMT* in more details.

**Batch Informed Tree (BIT*)**

The same as for informed RRT* (sec. 2.2.6), the word informed in the name “Batch Informed Tree” refers to the fact, that this motion planning algorithm knows when it found a solution from start to goal and uses this information for the further exploration of the configuration space.

3The notion of asymptotic optimality for FMT* is defined different than for sequential motion planning algorithms such as RRT*, informed RRT* and BIT*. FMT* is asymptotically optimal in the sense that it converges in probability to the optimal cost. This definition is weaker than the definition of asymptotic optimality in sec \[24\] used for RRT*, informed RRT* and BIT*. Janson et al. \[19\] explain asymptotic optimality for FMT* in more detail.
Figure 2.5: The FMT* generates a tree by steadily moving outward in the cost-to-arrive space. The exploration of the space by FMT* after 100 edges (left), 1000 edges (middle) and 2500 edges (right). Only edges are shown, samples are hidden. [19]

Algorithm 6 Fast Marching Tree algorithm (FMT*) [19]

1. **procedure** FMT*(requires a sample set $V$ comprised of $q_I$ and $n$ samples in $C_{free}$, at least one of which is also in the goal region $C_G$)
2. Place $q_I$ in $V_{open}$ and all other samples in $V_{unvisited}$; initialize tree with root node $q_I$
3. Find lowest-cost node $z$ in $V_{open}$
4. for each of $z$’s neighbors $q$ in $X_{unvisited}$ do
5. Find neighbor nodes $y$ in $V_{open}$
6. Find locally optimal one-step connection to $q$ from among nodes $y$
7. If that connection is collision-free, add edge to tree of paths
8. end for
9. Remove successfully connected nodes $q$ from $V_{unvisited}$ and add them to $V_{open}$
10. Remove $z$ from $V_{open}$ and add it to $V_{closed}$
11. Repeat until either
   (1) $V_{open}$ is empty $\Rightarrow$ report failure
   (2) Lowest-cost node $z$ in $V_{open}$ is in $C_G$ $\Rightarrow$ return unique path to $z$ and report success
12. **end procedure**
• BIT* is a probabilistically complete motion planning algorithm.
• BIT* is an asymptotically optimal motion planning algorithm. Gammell et al. [22] presented BIT* assuming that the optimization objective is path length. Hence, BIT* only works when the optimization objective is path length.
• BIT* does not consider the dynamics of a system.

The high-level procedure of BIT* is as follows: BIT* iteratively draws batches of samples. The batches of samples define an implicit random geometric graph (RGG). Every batch of samples is processed by running a graph search algorithm using a heuristic in order to find the shortest path from start to goal in the current batch of samples. An example of a heuristic is to choose the vertex in the tree with the lowest cost-to-go for further expansion of the graph search algorithm. Intuitively, this corresponds to search along a straight connection between start and goal first. The search results in a tree, whereas one branch corresponds to the shortest path, if it was found in the current batch of samples already. When a batch of samples is processed, a new batch of samples is added defining a denser implicit RGG. The graph search algorithm is run anew using the information from the last batches of samples, in order to update the tree and improve the current best path. Gammell, Srinivasa and Barfoot [22] explain BIT* in more details.

Kinodynamic Planning by Interior-Exterior Cell Exploration (KPIECE)
KPIECE is specifically designed for systems with complex dynamics and high-dimensional challenges.
• KPIECE is a probabilistically complete motion planning algorithm.
• KPIECE is not asymptotically optimal.
• KPIECE does consider the dynamics of a system.

The main idea of KPIECE is to sample control inputs to a dynamic system. With the transitions resulting from the inputs, a tree of motions is constructed. At each iteration, KPIECE selects a motion from the tree. Using a random control input, KPIECE expands the tree from a state along the selected motion using the sampled control input. Finally, information is extracted from the expansion process which will be used to expand the tree in the next iteration. Sucan et al. [13] explain KPIECE in more details.

2.3 Optimality
Trajectories can have different objectives. Examples of possible optimization objectives are
• Minimum path length trajectories / shortest paths (eq. 2.4 for $\mathbb{R}^3$)
• Time-optimal trajectories
• Energy-optimal trajectories
• Highest information gain trajectories
Algorithm 7 KPIECE Algorithm for a given initial configuration $q_I$ the algorithm proceeds for $N_{iterations}$. \[13\]

1: procedure KPIECE($q_I$, $N_{iterations}$) 
2: Let $\nu_0$ be the motion of duration 0 containing solely $q_I$ 
3: State $T = INITIALIZE_TREE(\nu_0)$ 
4: for $i \leftarrow 1$ to $N_{iterations}$ do 
5: $\nu = SELECTMOTION(T)$ 
6: EXPAND_TREE($T, \nu$) 
7: if solution is found then 
8: return solution 
9: end if 
10: EVALUATE_PROGRESS() 
11: end for 
12: return no solution 
13: end procedure

Figure 2.6: A tree of motions shown in the configuration space constructed by the KPIECE planner. \[13\]
As mentioned in sec. 1.2, during all of this thesis, path length is considered as optimization objective (eq. 2.4).

\[
L(\tilde{x}_{t_F}) = \int_0^{t_F} \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2 + \left(\frac{dz}{dt}\right)^2} \, dt,
\]

where \(\tilde{x}_{t_F}\) is a state trajectory in \(\mathbb{R}^3\), \(x, y, z\) are the \(x\)-, \(y\)- and \(z\)-component of \(\tilde{x}_{t_F}\), respectively.

Assuming no wind and constant energy consumption per unit of length shortest paths are equivalent to minimum-energy paths for aerial vehicles.

### 2.4 Collision checking

Collision checking assumes that the map is known.

A collision checking algorithm has to provide the functionality of determining if a state \(x\) is in collision \((x \in X_{\text{free}})\). In order to check whole paths for collision, the path is discretized into multiple states using a user-defined resolution. Subsequently, all states making up the path are checked for collision. This is done by implementing a function

\[
\text{bool collide(state) \{ return does\_state\_yield\_collision \}}
\]

which is called for every state considered to be passed when flying from start to goal. Often, collision checking algorithms only handle collision checking for the obstacle region \(O\). In other words, they only check the position for collision \(((x, y, z)^T \in W \setminus O)\). They do not check if the entire configuration is collision free \((q \in C_{\text{free}}\), all kinematic constraints), and neither if the state space constraints of a system are regarded \((x \in X_{\text{free}}\), e.g. constraints on the velocity).

#### 2.4.1 Flexible Collision Library

The Flexible Collision Library (FCL) is a C++ library to handle collisions between different collision objects.

FCL can check the whole configuration of a robot for collision \((q \in C_{\text{free}})\). FCL provides full three-dimensional collision checking with arbitrary maps and many other features:

- Collision objects such as the map or aerial vehicles can be represented in different ways and shapes. E.g. as a box, a sphere, an ellipsoid, as an arbitrary octomap or an arbitrary mesh.
- Distances between pairs of collision objects can be computed.
- In case of collision, FCL provides information including contact points and contact normals.

Pan, Chitta and Manocha [23] present FCL in more detail.

#### 2.4.2 Height checking

Height checking is a collision checking algorithm for the use with 2.5D maps (sec. 2.1). Height checking only checks if the position of the robot is obstacle free \(((x, y, z)^T \in W \setminus O)\).
Every cell of the 2.5D map, which is covered (also if only partly covered) by the projection of the collision object representing the airplane on the x-y plane, is checked for collision. This is done by checking the height of the map at this cell and the height of the state of the aerial vehicle. If for every covered cell, the height of the aerial vehicle is above the height of the map (e.g. including some safety distance), the state of the aerial vehicle is collision-free. Otherwise, the state of the aerial vehicle is in collision. The procedure is described in alg. 8.

Algorithm 8 Height checking.

1: procedure HEIGHT_CHECKING(2.5D map, x, shape and extent of the collision object at state )
2: Get the set of all cells C of the 2.5D map, which are covered (also partly) by the projection of the collision object on the x-y plane.
3: for c ∈ C do
4: if Is the height of the map in the cell c (possibly including a safety distance Δ) is above the height of the state of the aerial vehicle: c.z + Δ > x.z then
5: Return collision
6: end if
7: end for
8: Return State x is collision-free
9: end procedure

The computational complexity of the height checking algorithm is scalable by changing the resolution of the map.

2.5 System dynamics

In this section, an obstacle-free world is assumed (\( O \) is empty).

A dynamical system can often be modeled with a system of differential equations (eq. 2.5).

\[
\dot{x}(t) = f(x(t), u(t)) , \text{ where} \\
x(t) \in \mathbb{R}^n : \text{the vector containing the n state variables of a system} \\
u(t) \in \mathbb{R}^m : \text{the vector containing the m input variables of a system}
\]

In the following, two systems which are important for this thesis are described, the Dubins car and the Dubins airplane.

2.5.1 Dubins car

The Dubins car motion model [15] is expressed through

\[
\begin{align*}
\dot{x} &= V \cos(\Theta) \\
\dot{y} &= V \sin(\Theta) \\
\dot{\Theta} &= u
\end{align*}
\]

where

\( x, y \) is the position of the car
\( \Theta \) is the orientation of the car

\[
\mathbf{q} = \begin{pmatrix} x \\ y \\ \Theta \end{pmatrix} \in \mathcal{C} = \text{SE}(2) \text{ is the configuration of the car}
\]

\( V \) is the constant velocity of the car

\( u \) is the bounded input to the system \( u \in [\tan(\Phi_{\text{max}}), \tan(\Phi_{\text{max}})] \)

\( \Phi_{\text{max}} \) is the maximum steering angle

Intuitively, eq. 2.6 - 2.8 describe movements of a car which drives with a constant velocity (only in forward direction) on the x-y plane. The paths consist of straight lines and turns. The turns can have different turning radii, however there exists a minimum turning radius (eq. 2.9).

\[
\rho_{c,\text{min}} = \frac{V}{\tan(\Phi_{\text{max}})} \quad (2.9)
\]

The Dubins car has the property that the shortest feasible path between two configurations \((x_1, y_1, \Theta_1)^T\) and \((x_2, y_2, \Theta_2)^T\) is known in closed-form (sec. 2.5.1).

**Definition 1** Dubins car path

A Dubins car path is a shortest path connecting two configurations, \((x_1, y_1, \Theta_1)^T\) and \((x_2, y_2, \Theta_2)^T\), feasible for the Dubins car motion model.

**Dubins car paths**

Dubins [15] showed that the shortest path connecting any two configurations consists of straight lines (S) and turns with minimum turning radius \(\rho_{c,\text{min}}\) (L for left turn, R for right turn). Example paths of the Dubins car are shown in fig. 2.7. The shortest path between two configurations is always one of the six types: RSR, RSL, LSR, LSL, RLR, LRL. For arbitrary two configurations, a closed-form solution for the shortest path connecting them is known. Thus, the 6 path types can be computed efficiently. A straightforward procedure to compute the Dubins car path between two configurations is to compute the length of all six path types (RSR, RSL, LSR, LSL, RLR, LRL) and choose the shortest one. In order to speed up the computation of the Dubins car path between two configurations, Shkel and Lumelsky [4] developed a logical classification scheme, with which the path type of the Dubins car path connecting the two configurations can be determined with the information from the start and goal configuration. This avoids computing the length of all six path types.

**2.5.2 Dubins airplane**

The Dubins airplane is an extension of the Dubins car (sec. 2.5.1) to three dimensions. The Dubins airplane motion model [16] is expressed through

\[
\dot{x} = V \cos(\Theta) \\
\dot{y} = V \sin(\Theta) \\
\dot{z} = u_z \\
\dot{\Theta} = u_\Theta
\]

(2.10) 
(2.11) 
(2.12) 
(2.13)
(a) Shortest path between two configurations for the Dubins car. This path is a left, straight, left (LSL) type path.

(b) Shortest path between two configurations for the Dubins car. This path is a left, straight, right (LSR) type path.

(c) Shortest path between two configurations for the Dubins car. This path is a left, right, left (LRL) type path.

Figure 2.7: Dubins car paths

where

$x, y, z$ is the position of the airplane

$\Theta$ is the yaw of the airplane

$q = \begin{pmatrix} x \\ y \\ z \\ \Theta \end{pmatrix} \in C = \mathbb{R}^3 \times \text{SO}(2)$ is the configuration of the airplane

$V$ is the constant velocity relative to the ground

$u_z$ is the bounded climb and sink rate, $u_z \in [-u_{z,max}, u_{z,max}]$

$$\gamma = \arctan \left( \frac{u_z}{V} \right)$$ (2.14)

$\gamma$ is the bounded climb and sink angle (path angle)

$\gamma_{max} = \arctan \left( \frac{u_{z,max}}{V} \right)$ is the maximum climb and sink angle

$$u_{\Theta} = \frac{g}{V} \tan (\Phi)$$ (2.15)

$u_{\Theta}$ is the bounded turning rate, $u_{\Theta} \in [-\frac{g}{V} \tan (\Phi_{max}), \frac{g}{V} \tan (\Phi_{max})]$ 

$\Phi_{max}$ is the maximum bank angle

$g$ is the gravitational acceleration

Intuitively, eq. eq. 2.10 - 2.13 describe movements of an airplane which flies with a constant velocity in $\mathbb{R}^3$ in a certain direction given by the yaw (in the Dubins airplane motion model, roll and pitch are always zero). The paths consist of straight
The turns can have different turning radii, however there exists a minimum turning radius (eq. 2.16).

\[ r_{\text{min}} = \frac{V^2}{\tan(\Phi_{\text{max}}) \cdot g} \]  

(2.16)

In eq. 2.15 the algebraic relationship between the roll angle and the yaw is stated according to the coordinated turn condition described by Beard and McLain [24]. The climb or sink angle is bounded by \( \gamma_{\text{max}} \).

The Dubins airplane has the property that for any two configurations \((x_1, y_1, z_1, \Theta_1)^T\) and \((x_2, y_2, z_2, \Theta_2)^T\), the shortest path connecting them and feasible for the Dubins airplane motion model is known. For many two configurations it is also known in closed-form (sec. 2.5.1).

**Definition 2** Dubins airplane path

A Dubins airplane path is a shortest path connecting two configurations, \((x_1, y_1, z_1, \Theta_1)^T\) and \((x_2, y_2, z_2, \Theta_2)^T\), feasible for the Dubins airplane motion model.

In this thesis, the term *Dubins path* is used as a synonym for the term Dubins airplane path. Most often, optimal Dubins airplane paths is used instead of Dubins airplane paths to clarify the meaning.

**Dubins airplane paths**

The Dubins airplane motion model has a property that a solution for the shortest path between two states is known in obstacle free space. In many cases also a closed-form solution is known [16]. Note that shortest paths are equivalent to time-optimal paths since the Dubins airplane motion model assumes constant velocity. For this section, it is assumed that the space is obstacle-free.

It follows a summary of the work presented by Chitsaz and Lavalle [16]. In many cases, finding the shortest path between two configurations of the Dubins airplane can be reduced to finding the shortest path between the projections of the two configurations on the x-y plane for the Dubins car. For the Dubins car a closed-form solution for the shortest path connecting any two configurations is known. To find shortest paths for the Dubins airplane, three cases have to be considered separately. In two of the three cases, the shortest path for the Dubins airplane can be reduced to find the shortest path for the Dubins car and extend it to three dimensions.

Let \( q_1 = (x_1, y_1, z_1, \Theta_1)^T \) and \( q_2 = (x_2, y_2, z_2, \Theta_2)^T \) be the two configurations for which we want to know the shortest path for the Dubins airplane connecting them. Let \( q_{c,1} = (x_1, y_1, \Theta_1)^T \) and \( q_{c,2} = (x_2, y_2, \Theta_2)^T \) denote the projections of \( q_1 \) and \( q_2 \) on the x-y plane.

- **Low altitude case**: Assume the goal \( q_2 \) is at higher altitude than the start \( q_1 \). Imagine, the airplane flies exactly above the shortest path for the Dubins car \( p_c \) connecting \( q_{c,1} \) and \( q_{c,2} \). It flies with maximum climb angle \( \gamma_{\text{max}} \). If, at the end, the airplane is higher or at the same height as the goal state, the start-goal configuration is called low altitude.

    - It is shortest for the Dubins airplane to fly above the shortest Dubins car path \( p_c \) with an adequate climb angle \( \gamma = \arctan \left( \frac{z_2 - z_1}{p_c \text{ length}} \right) \). The same with maximum sink angle \(-\gamma_{\text{max}}\) if the goal is at lower altitude than the start.

A procedure to compute shortest path for the Dubins airplane for the low altitude case is given in alg. 9. 


• High altitude case: Assume the goal $q_2$ is at higher altitude than the start $q_1$. Imagine, first the airplane flies exactly above the shortest path for the Dubins car $p_c$ connecting $q_{c,1}$ and $q_{c,2}$. It flies with maximum climb angle $\gamma_{max}$. Subsequently, the airplane flies a 360° helix with minimum turning radius and maximum climb angle. If, at the end, the airplane is at the same height or below the height of the goal state, the start-goal configuration is called high altitude.

  - It is shortest for the Dubins airplane to fly above the shortest Dubins car path $p_c$ with maximum climb angle, followed by a helix with $N \cdot 360°$ with adequate turning radius $r_{opt} = \frac{|z_2 - z_1| - p_c.length() \cdot \tan(\gamma_{max})}{2 \pi N \cdot \tan(\gamma_{max})} \geq r_{min}$ and maximum climb angle $\gamma_{max}$. $N$ is the number of turns the airplane can fly with minimum turning radius such that its height at the end is below or equal to the height of the goal. The same with maximum sink angle $-\gamma_{max}$ if the goal is at lower altitude than the start.

A procedure to compute shortest path for the Dubins airplane for the high altitude case is given in alg. 10.

• Intermediate altitude case: Assume the goal $q_2$ is at higher altitude than the start $q_1$. Imagine, first the airplane flies exactly above the shortest path for the Dubins car $p_c$ connecting start $q_{c,1}$ and goal $q_{c,2}$. It flies with maximum climb angle. Assume, that at the end, the airplane is still lower than the height of the goal state $q_2$. Subsequently, the airplane flies a helix with 360°, minimum turning radius and maximum climb angle. Now, if the airplane is above the height of the goal state, the start-goal configuration is called intermediate height. This case can further be split up into two groups.

  - There exists a path with maximum climb angle. The shortest path connecting start and goal with a flyable path for the Dubins airplane is any path with maximum climb angle.

  - There does not exist a path with maximum climb angle. The shortest path connecting start and goal with a flyable path for the Dubins airplane is a path consisting of turns with minimum radius and straight line segments.

The same with maximum sink angle $-\gamma_{max}$ if the goal is at lower altitude than the start. For the intermediate case, to the best of my knowledge, no closed-form solution for the shortest path is known.

Algorithm 9 Procedure to compute the shortest path for the Dubins airplane motion model between two states $x_f, x_G$ for the low altitude case in obstacle-free space.

1: procedure COMPUTE_LOW_PATH($x_f, x_G, r_{min}$)
2: \hspace{1em} $dz \leftarrow x_f.z - x_G.z$
3: \hspace{1em} $p_{DubCar} \leftarrow \text{COMPUTE_DUBINS_CAR_PATH}(x_f.x, x_f.y, x_f.\Theta, x_G.x, x_G.y, x_G.\Theta, r_{min})$
4: \hspace{1em} $\gamma \leftarrow \arctan\left(\frac{dz}{p_{DubCar}.length()}\right)$
5: \hspace{1em} $p \leftarrow p_{DubCar}.\text{extendTo3D}(\gamma)$
6: Return $p$
7: end procedure
Algorithm 10 Procedure to compute the shortest path for the Dubins airplane motion model between two states $x_I, x_G$ for the high altitude case in obstacle-free space.

1: procedure COMPUTE_HIGH_PATH($x_I, x_G, r_{min}, \gamma_{max}$)  
2:  \[ dz \leftarrow x_I.z - x_G.z \]  
3:  \[ p_{DubCar} \leftarrow \text{COMPUTE_DUBINS_CAR_PATH}(x_I.x, x_I.y, x_I.\Theta, \ x_G.x, x_G.y, x_G.\Theta, r_{min}) \]  
4:  \[ N \leftarrow \text{floor} \left( \left( \frac{|dz|}{\tan(\gamma_{max})} - p_{DubCar}.\text{length}(\cdot) \right) \frac{1}{2 \pi r_{min}} \right) \]  
5:  \[ r_{opt} \leftarrow \frac{|dz| - p_{DubCar}.\text{length}(\cdot) \tan(\gamma_{max})}{2 \pi N \tan(\gamma_{max})} \]  
6:  \[ \gamma \leftarrow \text{sgn}(dz) \cdot \gamma_{max} \]  
7:  \[ p_{Helix} = \text{COMPUTE_HELIX}(N, r_{opt}, \gamma) \]  
8:  \[ p \leftarrow p_{DubCar}.\text{extendTo3D}(\gamma) \]  
9:  \[ p \leftarrow p.p.append(p_{Helix}) \]  
10: Return $p$  
11: end procedure

Dubins airplane quasimetric

The length of the Dubins airplane path as defined in def. 2 (assuming shortest paths also for the intermediate altitude case) defines a quasimetric. This is sometimes referred to the Dubins airplane distance. It is not a proper metric since it does not fulfill the symmetry condition required for metrics [6].

Limitations of the Dubins airplane motion model

The Dubins airplane motion model has two major limitations.

- The Dubins airplane motion model does not consider winds.
- The Dubins airplane paths have non-continuous curvature.

Winds may have a strong impact on the airplane. Due to high winds, some zones may become non-flyable. McGee, Spry and Hedrick [25] and Bakolas and Tsiotras [26] present tools to compute time-optimal paths in the case of constant wind. In order to exactly follow Dubins airplane paths, the airplane has to stop after each path segment, set the control unit from straight ahead to on full lock for a turn or the other way round, and continue flying. To avoid the unrealistic stopping in the air, the steering control unit would have to change the control input such that the control vane for the yaw changes infinitely fast. Computing paths with continuous curvature corresponds to finding shortest path for fixed-wing aerial vehicles, with the additional constraint of $\left| \frac{\dot{k}}{k} \right| < \sigma_{max}$, where $k$ is the curvature of a path and $\sigma_{max}$ the maximum change in curvature. Scheuer and Laugier [27], Skari et al. [28] and Richter, Bry and Roy [29] present methods to generate continuous curvature paths.
Chapter 3

Path planning method

As mentioned in sec. 1.2, the goal of this thesis is to develop a path planning framework for planning collision-free shortest paths, dynamically feasible for fixed-wing aerial vehicles. In addition, the path planning framework should be real-time capable. Since the planning runs onboard, all of this has to be achieved on a computer with low computational power.

The requirements for the path planning framework summarized:

• Find
  – collision-free and
  – dynamically feasible (approximated by Dubins airplane paths) and
  – minimum length
paths.

• Find an initial path in less than 1 second to ensure that the airplane knows any collision-free path to follow.

• Find an improved short path in about 2 seconds.

• Be compatible with most companion-computers installed on common fixed-wing aerial vehicles.

During the whole thesis, we made three assumptions:

• We want minimum length paths.

• There is no wind.

• The airplane can track paths with non-continuous curvature changes (paths in $C^1$).

Minimum length paths can be a good approximation for minimum energy paths. Under some assumptions it is equivalent to minimum energy paths (sec. 2.3). Since, this thesis is a first implementation, no wind is assumed for the sake of low computation times and simplicity. There are variants to consider wind, however, they are computationally more expensive [25], [26]. By choosing the airplane control-input limitations (maximum roll angle and maximum climb rate / maximum climb angle) conservatively, paths with non-continuous curvature changes can be tracked reasonably, and the computationally expensive process to compute continuous curvature paths is avoided.

The methods for the development of a path planning framework fulfilling the stated requirements under the stated assumptions are presented in the following sections.
3.1 Software structure

Contribution: The path planning framework is contained in a ROS [30] node implemented in C++. The framework provides the full range of functionalities for efficient path planning, path analysis and visualization and motion planner performance benchmarking (fig. 3.1). The interface and the structure for the path planning is provided by OMPL [12]. The structure of OMPL requires to define the motion planning setup consisting of start and goal, a map and a collision checking algorithm, a state space and an embedded metric, a procedure to compute a trajectory between two states, an optimization objective and a motion planning algorithm. Using these functionalities, OMPL handles the cooperation for planning shortest, collision-free and flyable paths from start to goal. For path analysis and visualization and planner performance benchmarking, we implemented further functionalities.

The key features of the path planning framework are visualized in fig. 3.1 and described below:

- Optimality objective: We used the path length optimization objective provided by OMPL.
- Collision checking: We implemented collision checking with FCL and also a collision checking method based on an own height checking algorithm (sec. 3.2).
- Airplane motion model: We implemented the state space according to the Dubins airplane motion model and defined a distance function and a procedure to compute trajectories between two states (sec. 3.3).
- Path planning algorithm: We used several path planning algorithms provided by OMPL. We modified some of them to improve their performance (sec. 3.4).
- Evaluation: We implemented functionalities to benchmark the path planning algorithms (sec. 3.5).
- Visualization: We implemented functionality to visualize the resulting path and intermediate results of the planning procedure (keep and visualize early solutions and the evolution of the tree of motions).
- Mission parameters: We provided various parameters accessible during runtime to modify the setup for planning, visualize intermediate results of the planning procedure and facilitate planner evaluation. The start and goal state are interactively modifiable whereas replanning is triggered on the reception of a new start or goal.

The framework handles all systems which can be modeled as a Dubins airplane [25.2]. These are mainly fixed-wing aerial vehicles. Furthermore, the framework handles only the minimum path length case.

3.2 Collision checking

For collision checking (check if the state \( x \) of the airplane is in \( X_{\text{free}} \)), we assumed that the airplane is a cube with a side length which is airplane dependent (maximum extent of the airplane, usually wing-span plus some distance for safety reasons).

\(^{1}\)It is possible to use generalized metrics (such as a quasimetric or a premetric). In this case, for some motion planning algorithms properties like probabilistic completeness and asymptotic optimality cannot be guaranteed anymore.
Figure 3.1: The software structure of the path planning framework. Inputs to the framework are the map, the start and goal state and various design parameters. The output of the framework is a path consisting of states (position and orientation).

Two different collision checking methods are implemented in the path planning framework.

- A method leveraging the C++ based Flexible Collision Library (sec. 2.4.1)
- Height checking (sec. 2.4.2)

FCL provides flexible three-dimensional collision checking. Height checking is less flexible than FCL. It handles only 2.5D maps and in its current implementation, the fixed-wing aerial vehicle can only be modeled as a cube. Due to its scalability, height checking can be made much faster than FCL. We usually chose a map resolution of the same size as the cube representing the airplane. With this choice, for every state collision check only 4 values have to be checked. Using height checking, only a minor amount of time during the planning procedure is spent on collision checking.

### 3.3 Airplane motion model

To model the dynamics of fixed-wing unmanned aerial vehicles, we used the Dubins airplane motion model (sec. 2.5.2). The Dubins airplane motion model has the property that the shortest path between two states is known between any two start and goal configurations (sec. 2.5.2). For many start-goal configurations, the shortest path is known in closed-form (sec. 3.3.1)

For path planning the state space of the Dubins airplane is the same as the configuration space of the Dubins airplane $X = C$. This is due to the constant velocity of the Dubins airplane, and the fact that the shortest path between any two start and goal configurations is known. Any geometric motion planning algorithm can
efficiently be used to plan paths for the Dubins airplane motion model using Dubins airplane paths as connections between states (sec. 2.2.6). No closed-form solutions for the Dubins airplane path for the intermediate altitude case (sec. 2.5.2) have been presented until now. In order to avoid optimization procedures or complex case distinctions to compute optimal paths for the intermediate case, we decided to solve the intermediate altitude case non-optimally (sec. 3.3.1). All other paths are shortest. The computation of the non-optimal Dubins airplane paths (non-optimal only for intermediate altitude cases) and lengths of non-optimal Dubins airplane paths during planning is done as follows:

- Low altitude case: Alg. 9 returns shortest paths
- High altitude case: Alg. 10 returns shortest paths
- Intermediate altitude case: Alg. 11 does not return shortest paths

Definition 3 Non-optimal Dubins airplane path
A non-optimal Dubins airplane path is a path connecting two configurations, \((x_1, y_1, z_1, \Theta_1)^T\) and \((x_2, y_2, z_2, \Theta_2)^T\), feasible for the Dubins airplane motion model, computed with the algs. 9, 10, 11.

During this thesis, paths computed with algs. 9, 10, 11 are denoted as non-optimal Dubins airplane paths. Note that this is a very strong definition. Only intermediate altitude cases results in non-optimal paths. In realistic application scenarios, about 5% of all start-goal configurations are intermediate altitude cases (sec. 4.2.3). As defined in def. 2 Dubins airplane paths refer to paths which are shortest for every case. Conclusively, for the low and high altitude cases, Dubins airplane paths and non-optimal Dubins airplane paths are the same.

3.3.1 Non-optimal Dubins airplane paths
The low altitude and the high altitude case are solved optimally in the path planning framework. Contribution: In order assure that collision-free flyable paths are found quickly, we decided to solve the intermediate altitude case non-optimally (sec. 2.5.2). The paths are computed as described in alg. 11

Algorithm 11 Algorithm to find a not necessarily optimal path for the intermediate altitude case, flyable for the Dubins airplane.

1: procedure COMPUTE_INTERMEDIATE_PATH\((x_f, x_G, r_{min}, \gamma_{max})\)
2: \(dz \leftarrow x_f.z - x_G.z\)
3: \(p_{DubCar} \leftarrow COMPUTE_DUBINS_CAR_PATH(x_f.x, x_f.y, x_f.\Theta, x_G.x, x_G.y, x_G.\Theta, r_{min})\)
4: \(N \leftarrow \text{floor}\left(\left|\frac{dz}{\tan(\gamma_{max})} - p_{DubCar}.\text{length()}\right| * \frac{1}{2*\pi*r_{min}}\right) + 1\)
5: \(\gamma \leftarrow \arctan\left(\frac{dz}{p_{DubCar}.\text{length()} + N*2*\pi*r_{min}}\right)\)
6: \(p_{Helix} \leftarrow COMPUTE_HELIX(N, r_{min}, \gamma)\)
7: \(p \leftarrow p_{DubCar}.\text{extendTo}3D(\gamma)\)
8: \(p \leftarrow path.append(p_{Helix})\)
9: Return \(p\)
10: end procedure

Some remarks on the quantification of the non-optimality:

Remark 1 Obstacle-free three-dimensional space: Using alg. 11, the non-optimality of the resulting path between two states in the intermediate altitude case can be upper-bounded by eq. 3.1.
\[ p_{\text{subopt}}.\text{length}() - p_{\text{opt}}.\text{length}() < \frac{2 \pi r_{\text{min}}}{\cos(\gamma_{\text{max}})} \]  

(3.1)

**Remark 2** All optimal paths (in arbitrary maps) which do not contain any intermediate altitude case path are still connected optimally. For such paths, probabilistically complete and asymptotically optimal planners stay probabilistically complete and asymptotically optimal.

**Remark 3** When considering the non-optimal paths, the length of non-optimal Dubins airplane paths does not define a quasimetric anymore (sec. 2.5.2) since the triangular inequality is not satisfied. The length of non-optimal Dubins airplane paths defines a premetric (the proof is in sec. A.2).

Rem. 1 was also verified experimentally (sec. 4.2.3). This requires the computation of (optimal) Dubins airplane paths. We implemented the procedure described by Beard, McLain and Timothy [31] for the computation of Dubins airplane paths in the intermediate altitude case. This procedure has some relevant disadvantages:

- The procedure can only be applied to long path case.\(^2\)
- The procedure does not return the optimal path for all start-goal configurations, even in the long path case (sec. 5.2).
- The procedure is computationally more expensive than computing non-optimal paths as described in alg. 11 since it involves an iterative optimization (sec. 4.2.3).
- The quality of the resulting paths depends on the step size of the optimization and therefore deviates from the truly optimal Dubins airplane path (sec. 4.2.3).

3.3.2 Dubins set classification

Computing (non-optimal and optimal) Dubins airplane paths requires the computation of Dubins car paths. In order to compute (non-optimal and optimal) Dubins airplane paths fast, it is required that the computation of Dubins car paths is fast. For fast computation of Dubins car paths a logical classification scheme can be applied. The classification scheme presented by Shkel and Lumelsky [4] is implemented in the path planning framework for long path cases\(^3\) and used for the computation of the non-optimal Dubins airplane paths.

3.3.3 Dubins airplane state space for OMPL

As mentioned in sec. 3.1 for the interface of the path planning software, OMPL [12] was used. OMPL provides a Dubins car state space. The Dubins car state space is a C++ class which allows to do path planning for the Dubins car. It implements the functionality of computing Dubins car paths and the corresponding path lengths. **Contribution:** We extended this class to four dimensions \((x, y, z, \Theta)\)^\(^4\) and adapted it to the Dubins airplane motion model. This class allows to do motion planning for the Dubins airplane (fig. 3.2). The class is called DubinsAirplaneStateSpace.

\(^2\)Do not confuse the definition of short and long path case (sec. 2.1) with the definition of low, intermediate and high altitude case (sec. 2.5.2). Start and goal goal states between which the horizontal distance is large are called long path cases.

\(^3\)For the definition of long path case consider [4]. It is a definition for the Dubins car. Roughly, it describes the case where start and goal are far enough from each other such that a RSR, RSL, LSR or LSL type path is optimal.
it defines the state (or configuration since $X = \mathcal{C}$) of the airplane. It computes non-optimal Dubins airplane paths and defines the distance function for computing distances between two states as the length of the non-optimal Dubins airplane paths computed with algos. [9] [10] [11].

Figure 3.2: Shortest non-optimal Dubins airplane paths planned using the implementation of the Dubins airplane state space. Note that these paths are optimal concerning path length since none of them contains an intermediate altitude case path.

3.4 Path planning

The path planning framework provides the use of nine different sampling-based path planning algorithms for fixed-wing unmanned aerial vehicles (summarized in tab. 3.1).

- 1 control-based planning algorithm (tab. 2.1) is implemented.
  - KPIECE

- 8 control-based planners using the fact that the closed-form solution for the shortest path for the Dubins airplane between two states is known for many states.
  - 2 planners which can handle arbitrary optimization objectives.
    - D-RRT*
    - D-FMT*
  - 2 planners which exploit the fact that we search for shortest paths and hence only work for minimum length paths.
    - D-IRRT*
    - D-BIT*

RRT* consecutively samples one state at a time, whereas BIT* draws batches of samples. For both planners, 2 additional modifications were implemented

  - DA-IRRT*
  - CA-IRRT*
  - DA-BIT*
  - CA-BIT*

The 9 planners briefly explained:

- **KPIECE** was already implemented. Inputs of the airplane are randomly sampled, and then knowing the ODEs describing the airplane, the system is forward propagated for a user-defined amount of time (sec. 2.2.6 and 2.2.4).
Table 3.1: Summary of the control-based path planning algorithms available in the path planning framework

<table>
<thead>
<tr>
<th>Planning category</th>
<th>Forward propagation (sec. 2.2.4)</th>
<th>Solve BVP (sec. 2.2.4)</th>
<th>Optimization objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>Non-optimal</td>
<td>Arbitrary</td>
<td>Path length</td>
</tr>
<tr>
<td></td>
<td>KPIECE</td>
<td>D-RRT*</td>
<td>D-IRRT*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DA-IRRT*</td>
<td>CA-IRRT*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D-FMT* (modified)</td>
<td>D-BIT*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DA-BIT*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CA-BIT*</td>
</tr>
</tbody>
</table>

- **D-RRT** uses the **RRT** motion planning algorithm (sec. 2.2.6) whereas non-optimal Dubins airplane paths are used for the connections between states (sec. 2.2.4). Distances between states are defined to be the length of the path connecting them.

- **D-IRRT** uses the informed **RRT** motion planning algorithm (sec. 2.2.6) whereas non-optimal Dubins airplane paths are used for the connections between states (sec. 2.2.4 and 3.4.2). Distances between states are defined to be the length of the path connecting them.

- **DA-IRRT** is a modification of D-IRRT* exploiting the fact that the dynamical system is an airplane and therefore adapts the informed subset on which the search for better paths is focused (sec. 3.4.3).

- **CA-IRRT** uses a modified euclidean distance metric (eq. 3.4) in order to find an initial path as fast as possible. Knowing the initial solution, the planning is continued using DA-IRRT*. Hence, CA-IRRT* combines two distance functions, a modified euclidean distance and the distance defined as the length of non-optimal Dubins airplane paths (sec. 3.4.4).

- **D-FMT** uses the **FMT** motion planning algorithm (sec. 2.2.6) whereas non-optimal Dubins airplane paths are used for the connection between states (sec. 2.2.4). Distances between states are defined to be the length of the path connecting them.

- **D-BIT** uses the **BIT** motion planning algorithm (sec. 2.2.6) whereas non-optimal Dubins airplane paths are used for the connections between states (sec. 2.2.4). Distances between states are defined to be the length of the path connecting them.

- **DA-BIT** is a modification of D-BIT* exploiting the fact that the dynamical system is an airplane and therefore adapts the informed subset on which the search is focused (sec. 3.4.3).

- **CA-BIT** uses a modified euclidean distance metric (eq. 3.4) in order to find an initial path as fast as possible. Knowing the initial solution, the planning is continued using DA-BIT*. Hence, CA-BIT* combines two distance functions, a modified euclidean distance and the the distance defined as the length of non-optimal Dubins airplane paths (sec. 3.4.4).
Furthermore, the framework contains a geometric planner, using the euclidean metric and straight lines as connections, for comparisons.

### 3.4.1 Control-based planning

To implement a control-based planner based on sampling an input and forward-propagating the system is the most straight-forward approach to implement in OMPL when dealing with a dynamical system. The dynamical system has to be representable as a system of ordinary differential equation. The planning algorithm randomly samples an input and forward propagates the dynamical system with a user-defined time-step (KPIECE). This was the baseline solution that was extended during this thesis.

As mentioned in sec. 2.2.4, control-based planning with a forward propagation routine does not provide optimality guarantees and is computationally more expensive (fig. 3.3).

![Figure 3.3: The path found using control-based planning with a forward-propagation routine (left) and control-based planning knowing the closed-form solution to the BVP using a geometric motion planning algorithm (right) after 0.3 seconds in the same setup. KPIECE (left) or any other planning algorithm based on a forward propagation routine is slower than control-based planning with the closed-form solution such as DA-IRRT* (right).](image)

In order to conceptually have a guarantee on asymptotically optimal paths and to speed up the planning procedure, eight further control-based planning algorithms mentioned at the beginning of sec. 3.4 were added in the path planning framework. All of these eight path planning algorithms are specifications of alg. 4. The specifications for alg. 4 are as follows:

- All eight planners use an underlying geometric motion planning algorithm \( \mathcal{A} \) described in sec. 2.2.6 (RRT*, informed RRT*, FMT*, BIT*).
- According to the Dubins airplane, the states consist of position and orientation, \( x = (x, y, z, \Theta)^T \). Also, \( X = \mathbb{R}^3 \times \text{SO}(2) = \mathbb{C} \).
- While planning with \( \mathcal{A} \), for each sample all components of the state \( x = (x, y, z, \Theta)^T \) are randomly sampled.
- While planning with \( \mathcal{A} \), all distance queries (e.g. for nearest neighbor queries) are processed by computing the length of the non-optimal Dubins airplane path (used in step 4, 5 and 6 of alg. 4). This requires the computation of non-optimal Dubins airplane paths. An exception is the search of an initial solution with CA-IRRT* and CA-BIT*, where eq. 3.4 is used instead.
• While planning with $A$, all edges in the graph constructed by $A$ are non-optimal Dubins airplane paths (used in step 6 of alg. 4). This requires the computation of non-optimal Dubins airplane paths.

An exception is the planning of an initial solution with CA-IRRT* and CA-BIT*, where edges are straight lines instead.

• While planning with $A$, every non-optimal Dubins airplane path is computed from the closed-form solution to the BVP for the Dubins airplane presented in algs. 9, 10 and 11 (used in step 4, 5 and 6 of alg. 4). For the low and high altitude case, the solution from algs. 9, 10 is optimal with respect to path length. For the intermediate altitude case, the solution from alg. 11 is non-optimal with respect to path length.

• The collision checking is done using height checking or FCL (used in step 4, 5 of alg. 4).

### 3.4.2 Informed sampling

Recall that a basic assumption for this thesis is that the aim is to find minimum length paths. The ellipsoidal informed subset $X_c$ (for a system without differential constraints and for euclidean distance as cost function) contains only states for which, assuming obstacle free space, the straight-line path from start through any state in the ellipsoid to the goal is shorter or equal to the length of the current best path (every straight-line path from the start through a state on the surface of the ellipsoids to the goal has the same length as the length of the current best path). However, the path planning algorithms used, use non-optimal Dubins airplane paths as connections between states (sec. 3.4.1). The length of a straight-line connecting two states gives a lower bound on the length of the non-optimal Dubins airplane path between the two states. Due to this fact, $X_c$ overestimates the region of states which potentially can improve the current best path, in case of control-based planning using non-optimal Dubins airplane paths as connections. Still, all states that potentially can improve the current best solution for the Dubins airplane, $X_{c,D}$, are contained in the ellipsoid $X_c$, specified in fig. 2.2 ($X_{c,D} \subset X_c$).

Furthermore, since we are planning for the Dubins airplane, all components of its state ($x, y, z, \Theta$) are randomly sampled. Informed sampling only restricts the search for $x, y, z$, whereas the yaw $\Theta$ is not restricted. This makes informed sampling less efficient when planning for the Dubins airplane.

### 3.4.3 Dubins airplane adapted sampling

**Contribution:** Dubins airplane adapted sampling is an adaptation of informed sampling which exploits the dynamics of the Dubins airplane. Dubins airplane adapted sampling is deployed in the DA-IRRT*, CA-IRRT*, DA-BIT* and CA-BIT* motion planning algorithm.

For the Dubins airplane (a system with differential constraints) and path length as cost function), the informed subset $X_{c,D}$ is more complex. It can be represented by eq. 3.2

$$X_{c,D} = \left\{ x \in X \mid \text{DubinsAirplanePath}(x_I, x).\text{length()} + \text{DubinsAirplanePath}(x, x_G).\text{length()} \leq c_{\text{best}} \right\}$$
It is not possible to sample this region directly. It is computationally inefficient to do rejection sampling especially when the region is small compared to the whole state space. However, this more complex informed subset is completely contained in $X_{c,D} \subset X_c$. An intuitive approach to approximate $X_{c,D}$ more closely with $X_{\hat{c},D}$, is to cut off the top and the bottom of the ellipsoidal informed subset $X_c$ (fig. 3.4). The reasoning behind cutting off the top and the bottom of the ellipsoid is due to the fact that for flying up or down, the airplane has to spiral up or down which yields longer paths than flying to the left or right. The cut surfaces are always horizontal. Mathematically expressed, the approximate informed subset results in eq. 3.3.

$$X_{\hat{c},D} = \left\{ x \in X \mid \max \left\{ \left\| \begin{pmatrix} x_{I.x} \\ x_{I.y} \\ x_{I.z} \end{pmatrix} - \begin{pmatrix} x.x \\ x.y \\ x.z \end{pmatrix} \right\|_2^2 + \left\| \begin{pmatrix} x.x \\ x.y \\ x.z \end{pmatrix} - \begin{pmatrix} x_{G.x} \\ x_{G.y} \\ x_{G.z} \end{pmatrix} \right\|_2^2 \right. \\
\left. \quad \frac{|x_{I.z} - x.z|}{\sin(\gamma_{\text{max}})} + \frac{|x.z - x_{G.z}|}{\sin(\gamma_{\text{max}})} \right\} \leq c_{\text{best}} \right\}$$

(3.3)

Figure 3.4: The informed subset adapted to the Dubins airplane motion model.

This shape (fig. 3.4) contains all states which yield a shorter path than the current shortest path for the Dubins airplane. It still contains states that cannot improve the current shortest path since in general the non-optimal Dubins airplane path between two states is longer or equal to the length of the straight-line connection. Nevertheless, the ellipsoid $X_c$ is computed as if we were using straight-line connections (fig. 2.2). However, due to cutting off the top and the bottom of the ellipsoid, the amount of states which cannot improve the solution is reduced ($X_{c,D} \subset X_{\hat{c},D} \subset X_c$).

### 3.4.4 Combining the use of straight lines and non-optimal Dubins airplane paths

**Contribution:** The motivation behind additionally using straight lines for the path planning is that for the airplane it is desirable to find an initial path fast. This reduces the risk of crashing in obstacles due to long times without knowing any collision-free path. Combining the use of straight lines and non-optimal Dubins airplane paths is deployed in the CA-IRRT* and CA-BIT* motion planning algorithm. The idea behind this approach is that it is computationally less expensive to compute euclidean distances than lengths of Dubins airplane paths (non-optimal and optimal Dubins airplane paths). Euclidean distances can be seen as an approximation to non-optimal and optimal Dubins airplane paths which always underestimate the true length of a path.
• If start and goal are far from each other in horizontal direction compared to the minimum turning radius, and the vertical distance is adequately small, the euclidean distance is a good approximation for the length of the non-optimal and optimal Dubins airplane path.

• If start and goal are close to each other compared to the minimum turning radius, and the vertical distance is adequately small, the euclidean distance is a bad approximation for the length of the non-optimal and optimal Dubins airplane path in many cases.

• If the vertical distance is large compared to the horizontal direction, the euclidean distance is a bad approximation for the length of the non-optimal and optimal Dubins airplane path. This is due to fact that the airplane has to fly spirals in order to gain or lose height on a small horizontal extent.

In order to improve the approximation in the last case, instead of computing distances with the euclidean distance, eq. 3.4 is used.

\[
d(x_1, x_2) = \max \{d_{\text{eucl}}(x_1, x_2), d_{\text{DubinsApprox}}(x_1, x_2)\}
\]  (3.4)

\[
d_{\text{eucl}}(x_1, x_2) = \left\| \begin{pmatrix} x_{1,x} \\ x_{1,y} \\ x_{1,z} \end{pmatrix} - \begin{pmatrix} x_{2,x} \\ x_{2,y} \\ x_{2,z} \end{pmatrix} \right\|_2
\]  (3.5)

\[
d_{\text{DubinsApprox}}(x_1, x_2) = \frac{|x_{1,z} - x_{2,z}|}{\sin(|\gamma_{\text{max}}|)}
\]  (3.6)

Remark 4 Eq. 3.6 is an approximation to the length of the optimal Dubins airplane paths. For all optimal Dubins airplane paths with maximum path angle, eq. 3.6 computes the exact length of the path.

Remark 5 With the realistic assumption of \(\gamma_{\text{max}} \in (0, \pi]\), eq. 3.4 defines a pseudometric (the proof is in sec. A.1).

For the search of an initial solution, eq. 3.4 is used for computing distances between sampled states (mainly used for finding nearest neighbors, for further details see sec 2.2). Since eq. 3.4 does not compute the true distance between the two states for the airplane, the quality of the initial path may be bad. However, the time needed to find the initial solution is reduced.

As soon as an initial solution has been found, it is favored to rather find good quality paths than finding new paths quickly and maybe converge to a non-optimal path. Therefore, after an initial solution has been found, the planning is continued using the length of the non-optimal Dubins airplane path connecting two states as distance.

The straight-line paths are not flyable with the Dubins airplane due to the sharp corners which arise at the vertices (fig. 3.5). Therefore, as soon as the initial path with straight lines is found, the path consisting of straight-lines will be transformed into a path where the connections between the samples are non-optimal Dubins airplane paths. In order to assure that these paths will be collision-free, constantly during the construction of the graph, all straight-line connections are checked if there were a collision-free non-optimal Dubins airplane path.

3.4.5 Forward propagation

For the sake of compatibility with more complex airplane motion models different from the Dubins airplane (e.g. including aerodynamic effects and models for the energy source), the framework includes several control-based planners based on forward propagation routine (sec. 2.2.4):
3.4. Path planning

Figure 3.5: Planning an initial path by combining the use of straight lines and non-optimal Dubins airplane paths (taking the example of a sampling-based motion planning algorithm sampling one sample at a time). The tree of motions is shown after 1 (left), 3 (middle) and 4 (right, found an initial path) samples. Speed up for finding an initial path can be achieved by computing distances between samples using eq. 3.4 (e.g. for nearest neighbor queries). The straight-line path (solid lines in black) is not flyable by the Dubins airplane due to the sharp corners at the vertices. In order to assure that there exists a collision-free flyable path, constantly during the construction of the graph, every connection is checked for collision along the non-optimal Dubins airplane path between the two states (dashed in black).

- RRT
  - KPIECE
  - RRT modified for the usage with sampling controls and forward propagation (sec. 2.2.4).
  - Path-Directed Subdivision trees (PDST), Syclop[RRT], Syclop[EST] (Ladd [32] and Plaku [33] give more details about PDST and Syclop).

3.4.6 Optimal fast marching tree using non-optimal Dubins airplane paths (D-FMT*)

FMT* as presented in sec. 2.2.6 is inflexible. Depending on the user-defined amount of samples which are drawn and the computer used for planning, after a certain time $t_F$, FMT* finds a path (if one exists using the given samples). Before this time $t_F$ no path is known. Furthermore, the path is not improved anymore. This does not meet the requirements of a real-time capable motion planning algorithm (see the beginning of sec. 3). In addition, FMT* only satisfies a weaker definition of asymptotic optimality than RRT*, informed RRT* and BIT*. We added two modifications to FMT* in order to make it compatible with real-time requirements and to strengthen the optimality guarantees of FMT*. This resulted in two variations of FMT*.

Prediction of number of samples

*Contribution:* As described in sec. 2.2.6 FMT* uses a graph search algorithm to find the shortest path through the network implicitly produced by the samples. The complexity of graph search algorithms with respect to the amount of samples is known (e.g. for the Dijkstra algorithm, the time complexity is $t = k \ast (n \ast \log n + m)$, where $k$ is a computational power specific constant, $n$ is the number of vertices and $m$ is the number of edges in the graph [24]). Eq. 3.7 states the time complexity for FMT* which was proven by Janson et al. [19].
\( \mathcal{O}(n \cdot \log(n)) \), where
\( n \) : the number of vertices in the graph

(3.7)

Assuming that the time needed to find a path with \( N_1 \) samples is known and knowing the complexity of an algorithm, it is possible to predict the amount of samples \( N_2 \) which are workable in the remaining time available for planning. From the requirements stated at the beginning of sec. 3 the time available for planning is specified. An initial path has to be known before 1 second of planning, and a reasonably short path at about \( t_F = 2 \) seconds of planning. Choosing a rather small amount of samples for the first planning (e.g. \( N_1 = 300 \)), measuring the time needed to process \( N_1 \) samples \( t_{N_1} \), then predicting the amount of samples \( N_2 \) workable in the time remaining for planning \( t_{N_2} = t_F - t_{N_1} \) and replanning with \( N_2 \) samples, makes it possible to satisfy these requirements. The planning is not strictly interrupted after a certain amount of time \( t_F \) but finishes as soon as the second batch of \( N_2 \) samples is processed (approximately at about \( t_F \) seconds) allowing for flexibility concerning the run time. \( N_1 \) is chosen such that the time necessary for the first planning is much shorter than the total time available for planning. Under this assumption, \( N_2 \gg N_1 \), and hence the path resulting from the second planning is likely to be shorter than the first path. The procedure is described in alg. 12.

Algorithm 12 FMT* modified to predict the amount of samples which are workable in the time available for planning.

1: procedure FMT*_SAMPLE>PREDICTION(\( \mathbf{x}_I, \mathbf{x}_G, t_F, N_1 \))
2: Run FMT* as described in alg. 6 with \( N_1 \) samples and measure the time needed for planning \( t_1 \).
3: If no path was found with \( N_1 \) samples, set \( N_1 = 2 \cdot N_1 \) and go to step 1. Otherwise continue.
4: Commit the initial path \( p_1 \) to the airplane and continue.
5: if Time remaining for planning \( t_2 = t_F - t_1 > 0 \) then
6: Predict the amount of samples \( N_2 \) workable in \( t_2 \) from the complexity of the algorithm (eq. 3.7 or 3.8).
7: Rerun FMT* as described in alg. 6 with \( N_2 \) samples.
8: end if
9: If a second path \( p_2 \) was found and it is shorter than \( p_1 \), return \( p_2 \), otherwise return \( p_1 \).
10: end procedure

In order to improve the precision in prediction of the samples, the complexity of the algorithm was manually determined (sec. 4.2.2). FMT* was run 25 times for a different amount of samples. For each run the time for planning was measured. By fitting a polynomial curve to the set of data points the complexity of the algorithm was determined. The resulting relationship is shown in eq. 3.8

\[
t = k \cdot \left( \frac{n^2}{3500} + n \right), \quad \text{where}
\]
\[k : \text{a computational power specific constant}
\]
\[n : \text{the number of vertices in the graph}
\]

(3.8)

Compared to the straightforward extension of FMT* to make it anytime (sec. 3.4.6), the approach with sample prediction is more intelligent and resource-efficient. Only
a small amount of information ($N_1$ samples and associated information) is discarded. Additionally, due to the flexible run time, problems of missing an improved solution due to abrupt stopping after a specific time are avoided. The prediction of the amount of samples according to the complexity of the algorithm usually underestimates the true amount of samples workable in the time remaining for planning since the theoretical time complexity of an algorithm gives an upper bound on the real computation time. Therefore, an improved path will be known at the desired time $t_F$.

Resampling and replanning

Janson et al. [19] present an anytime variant of FMT*. The main idea of this variant is explained in alg. 13. This algorithm is implemented in the path planning framework without the reuse of information from previous iterations.

Algorithm 13 An anytime variant of FMT*.

1: procedure ANYTIME_FMT*(x_I, x_G, t_F, N_1)
2: Set $N = N_1$
3: while Time is up $t > t_F$ do
4: Run FMT* as described in alg. 6 with $N$ samples. Use information from the previous runs if available.
5: If a path was found, commit the path $p$ to the airplane if the path is better than the current best path, and continue.
6: Set $N = 2 \times N$
7: end while
8: end procedure

3.5 Manual code optimization

Contribution: In order to further speed up the planning of paths for fixed-wing aerial vehicles, we manually optimized the code. This mainly included the following changes:

- Precomputing trigonometric functions and exponentials
- Change if-else statements to switch statements
- Replace divisions with multiplications
- Pass large function parameters by reference or as pointers
- Postpone variable declarations until initialization, initialize and assign variables simultaneously

To build the code in release mode is not considered as manual code optimization. This resulted in an increase of the amount of iterations 86% in the same computation time (sec. 4.2.4).
Chapter 4

Experiments and results

Contribution: In order to evaluate the different planners, reveal their advantages and disadvantages and compare their overall performance, several experiments were done. Some measurements were collected using our own implementation to measure time data. Other measurements were collected using the benchmarking tools provided by OMPL [35]. Eight control-based path planning algorithms were evaluated (sec. 3.4). The motion planning algorithms based on forward propagation were not evaluated since they are evidently slower.

For the evaluation of the different planners itself, it is interesting to compare the length of the resulting path to the length of the shortest path for the Dubins airplane in the given map. For this, the length of the shortest path has to be known. In general, this value is not known. Hence, in order to approximately determine the shortest path, DA-IRRT* planned for 10 minutes and the resulting path was considered as optimal. This procedure can be justified when the planning is asymptotically optimal. As mentioned in sec. 3.3.1 this is not the case for our implementation in general. However, many of the experimental setups we used fall under the case of remark 2 and hence the procedure is justified (block experiment, gap experiment, random map experiment, slalom experiment). For the other experimental setups (mountain experiment, Kandervalley experiment and Kandervalley - restricted height experiment), the path which was found after 10 minutes of planning was inspected visually. The paths assumed to be optimal paths were inspected visually for approximate optimality and seemed reasonable for comparisons.

Four preliminary experiments presented in sec. 4.2 were done to test different aspects of the path planning framework. Seven main experiments presented in sec. 4.3 were conducted for the comparison of the eight planners. All experiments were run on a single core of an Intel Xeon 3.3GHz CPU. For all experiments, the airplane was assumed to be able to fly a minimum turning radius of $r_{min} = 25$ m and a maximum path angle of $\gamma_{max} = 0.15$ rad. Assuming a velocity of $9 \text{ m/s}$ these values correspond to a maximum climb and sink rate of $1.36 \text{ m/s}$ and a maximum roll angle of $18.3^\circ$. In order to be able to conduct a fair evaluation between the different planners, the parameters of all evaluated planners were tuned on two different maps with two different size scales (sec. 4.1).

4.1 Parameter tuning for planners

All evaluated planners provide different parameters for tuning. In order to assure fair comparison between the planners, all planners were tuned on the same maps. For some parameters, it makes sense to adapt them to the size scale of the map. For our experiments, maps with a side length of about 1km (block, gap, random,
slalom and mountain) and maps with a side length of about 10km (Kandervalley) were used. To cope with the size scale, two sets of parameters were chosen for each planner: One set of parameter for the small maps and one set of parameters for the large maps. Tab. 4.1 lists the parameters considered and a short description thereof. These were considered as most influential. For more details about the specific parameters, consider the OMPL documentation [12].

Table 4.1: Parameters considered for tuning

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Planners tunable with this parameter</th>
<th>Considered values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>The maximum length a motion can have. This parameter is expressed as a fraction of the space extent. Hence it scales automatically with the map size.</td>
<td>D-RRT*, D-IRRT*, DA-IRRT*, CA-IRRT*</td>
<td>space extent $^{2}$, space extent $^{5}$, space extent $^{10}$, space extent $^{40}$</td>
</tr>
<tr>
<td>Number of samples</td>
<td>Number of samples used for the first batch of samples (sec. 3.4.6). For this parameter, it may make sense to choose a bigger amount of samples for larger maps, if arguing to aim at having a certain density of sampled points in the map.</td>
<td>D-FMT*</td>
<td>300, 500, 1000</td>
</tr>
<tr>
<td>Samples per batch</td>
<td>Number of samples per batch of samples. Since the information gained from the previous batches of samples is reused, this parameter is map size independent.</td>
<td>D-BIT*, DA-BIT*, CA-BIT*</td>
<td>50, 100, 200, 500, 1000</td>
</tr>
</tbody>
</table>

The parameter tuning was conducted on the third map of the block experiment (sec. 4.3.1) and on the Kandervalley map using the first setup (sec. 4.3.6). For CA-IRRT* and CA-BIT*, the “range” and the “samples per batch” parameter could be set differently for the planning of the initial solution and the subsequent planning to improve the initial path resulting in more flexibility for the parameter tuning of CA-IRRT* and CA-BIT*. However, for both planners the same value was used in both steps.

The procedure to tune the parameters was as follows:

1. Do for both maps:
   (a) Run each planner for each parameter 100 times for 10 seconds.
   (b) Analyze each parameter for each planner
   (c) Neglect parameters for which some runs did not yield a solution.
   (d) Neglect parameters for which in more than 95% of the runs, the time needed to find an initial solution was more than 1 second.
   (e) Choose the parameter which yields the shortest path after 2 seconds.
   (f) In tight decisions, prefer parameters which yield an initial solution earlier and further look at the rate of convergence and smoothness of the convergence.
(g) For the parameter "number of samples", also consider the density of samples (samples per volume).

A map with an extent of 1500m x 1500m x 750m has a volume of 1.69km$^3$. A box with dimensions 200m x 200m x 100m has a volume of 4 $\times$ 10$^6$m$^3$. The horizontal dimensions of the box roughly correspond to sight distance with cameras. Assuming some obstacles in the map, 300 samples approximately yield a density of one sample per box. In order to have a similar density of samples per volume for large maps (12km x 12km x 5km), 128000 samples are needed. This is not tractable in reasonable time with common onboard computers. Hence, for large maps, an amount of samples was chosen for which in more than 95% of the runs, the time needed to find an initial solution is less than 1 second (number of samples = 500).

The resulting parameters are shown in tab. 4.2

Table 4.2: Parameters chosen for each planner for tuning

<table>
<thead>
<tr>
<th>Planner name</th>
<th>Block map</th>
<th>Kandervalley map</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-RRT*</td>
<td>range = $\frac{\text{space extent}}{10}$</td>
<td>range = $\frac{\text{space extent}}{10}$</td>
</tr>
<tr>
<td>D-IRRT*</td>
<td>range = $\frac{\text{space extent}}{10}$</td>
<td>range = $\frac{\text{space extent}}{10}$</td>
</tr>
<tr>
<td>DA-IRRT*</td>
<td>range = $\frac{\text{space extent}}{20}$</td>
<td>range = $\frac{\text{space extent}}{20}$</td>
</tr>
<tr>
<td>CA-IRRT*</td>
<td>range = $\frac{\text{space extent}}{20}$</td>
<td>range = $\frac{\text{space extent}}{20}$</td>
</tr>
<tr>
<td>D-FMT*</td>
<td>number of samples = 300</td>
<td>number of samples = 500</td>
</tr>
<tr>
<td>D-BIT*</td>
<td>samples per batch = 500</td>
<td>samples per batch = 1000</td>
</tr>
<tr>
<td>DA-BIT*</td>
<td>samples per batch = 500</td>
<td>samples per batch = 200</td>
</tr>
<tr>
<td>CA-BIT*</td>
<td>samples per batch = 500</td>
<td>samples per batch = 200</td>
</tr>
</tbody>
</table>

4.2 Preliminary results

4.2.1 FCL and height checking

This experiment was used to compare the computational cost coming with the two collision checking algorithms FCL and height checking.

Setup

We assumed that the airplane is a cube with a side length of 10m. The path planning framework was run for 100 seconds with DA-IRRT* on a random map (fig. 4.11). Four runs were done.

- Two runs using FCL, once with an octomap with a resolution of 10m and once with an octomap with a resolution of 2m.
- Two runs using height checking, once with a height map with a resolution of 10m and once with a height map with a resolution of 2m.
4.2. Preliminary results

Results

Changing the collision checking method from FCL to height checking reduced the time used for collision checking. Hence, there was more time available for other parts of path planning (fig. 4.1).

![Diagrams showing code time profile with height checking and FCL.]

- (a) Height checking with 2.5D maps, using a map resolution of 10m.
- (b) FCL with octomaps, using a map resolution of 10m.
- (c) Height checking with 2.5D maps, using a map resolution of 2m.
- (d) FCL with octomaps, using a map resolution of 2m.

Figure 4.1: Code time profile when running the code with height checking (left) and with FCL (right). The resolution of the 2.5D map as well as the octomap resolution was chosen to be 10m (top) and 2m (bottom). The time needed for collision checking is plotted in yellow. Furthermore, the portion of time needed for distance computations (green), interpolation between two states on a Dubins airplane path (red) and other computations (blue) is shown. The major part of other computations includes the (rejection) sampling procedure, tree pruning, bookkeeping of the vertices in the graph and checking if an improved solution was found and if a termination criterion or optimization objective is fulfilled.

Changing the resolution of the map affects the computation time for height checking with a quadratic relation. Increasing the resolution of the map by a factor of five, 25 times more time was spent on collision checking since 25 times more cells had to be checked for collision (25 \times 0.12\% = 3\% \approx 3.7\%, fig. 4.1). It could not be determined why FCL does not scale with different octomap resolutions.

Conclusively, this experiments suggests the use of height checking with a map resolution of 10m. This allows to assure that the position of the aerial vehicle is collision-free by checking at most 4 values while maintaining a reasonable accuracy.

---

1 An explanation could be that FCL scales both collision objects to the same resolution. The airplane is represented as a box of 10m x 10m x 10m, possibly, FCL uses the same resolution for the map collision object.
4.2.2 Determine the time complexity of FMT*

This experiment was conducted in order to determine the time complexity of FMT* experimentally. The reason for this experiment is explained in sec. 3.4.6.

**Setup**

FMT* was run 25 times for \( N = \{100, 300, 500, 1000, 1500, 2000, 4000, 6000\} \) in the third map of the block experiment (fig. 4.7). For each run, the time to find a path from start to goal was measured.

**Results**

The results of the experiment are shown in fig. 4.2.

![Experiment for determining the time complexity of FMT*](image)

Figure 4.2: Experiment for determining the time complexity of FMT*. The time needed to process \( N = \{100, 300, 500, 1000, 1500, 2000, 4000, 6000\} \) with FMT* in the third map of the block experiment is shown with blue circles (FMT* was run 25 times for each \( N \)). A polynomial of degree two (orange) is fitted in a least square sense (eq. 4.1).

The time complexity which resulted from this experiment is stated in eq. 4.1

\[
t = k \ast \left( \frac{n^2}{3500} + n \right), \text{ where}
\]

\[
k : \text{is a computational power specific constant}
\]

\[
n : \text{is the number of vertices in the graph}
\]

\[
t : \text{is the time needed to process the number of samples}
\]

4.2.3 Non-optimal vs. optimal Dubins airplane paths

This experiment aimed at experimentally verifying rem. 2 and examining the advantages and disadvantages of using non-optimal Dubins airplane paths instead of optimal Dubins airplane paths.
4.2. Preliminary results

Setup - path length and quality

The procedure presented by Beard and McLain [31] to compute optimal Dubins airplane paths can only be applied for long path cases. If the distance between start and goal state is more than $6 \cdot r_{\min}$ all start-goal configurations are long path cases.

The start state was set to $(x_I, x_I, y_I, x_I, z_I, x_I, \Theta)_T = (0, 0, 0, 0)$. One thousand goal states in $X = [-400, 400] \times [-400, 400] \times [-125, 125] \times [-\pi, \pi]$ were randomly sampled. Subsequently, for every goal state, the optimal Dubins airplane path and the non-optimal Dubins airplane path resulting from alg. 11 and their lengths were computed. The experiment was conducted in an empty map. It was done five times with different angle step sizes (for each angle step size, 1000 paths were computed). The procedure to compute optimal Dubins airplane paths is described in the work of Beard and McLain [31].

Results - path length and quality

In fig. 4.3 the results for 40 randomly selected goal states are shown. The optimal paths were computed with an angle step size of 5°. In fig. 4.4 the increase in computation time using optimal Dubins airplane paths and the mean deviation from the goal state, both only for intermediate altitude cases, are shown.

Figure 4.3: The results for 40 goal states randomly sampled out of the 1000 runs for an angle step size of 5° (all 1000 optimal paths were computed with a angle step size of 5°). The first plot shows the computation time needed to compute the optimal and non-optimal Dubins airplane paths. The second plot shows the length of the resulting paths. The third plot shows the difference in length of the non-optimal and the optimal paths. The last plot shows if the start-goal configuration is a low altitude, intermediate altitude or high altitude case.

- The median time needed to compute optimal Dubins airplane paths with an angle step of $d\alpha = 5^\circ$ was more than 10 times higher than the time needed to
Chapter 4. Experiments and results

Figure 4.4: These plots are computed from intermediate altitude cases only. Both values were computed with different angle step sizes in the optimization procedure which arises when computing optimal Dubins airplane paths for intermediate altitude cases. The first plot shows the median values for the ratio between the computation time needed for optimal Dubins airplane paths and non-optimal Dubins airplane paths. The second plot shows median deviation from the goal state of the last state of the the optimal Dubins airplane paths. This can be interpreted as a measure for the deviation from the truly optimal Dubins airplane path (sec. 3.3.1). The error bars denote the 5th and 95th percentile in both plots.

- compute non-optimal Dubins airplane paths with alg. 11
- The difference in length between the start state and the goal state is upper-bounded by eq. 3.1 which in our case was $\frac{2\pi \times 25 \cos(0.15)}{2} \approx 157$ m.
- Increasing the angle step size $d\alpha$ decreased the computation time but increased the deviation of the computed optimal path to the truly optimal path.

The experiment shows that computing optimal Dubins airplane paths with the procedure presented by Beard and McLain [31] is computationally more expensive than using our alg. 11 for computing non-optimal paths. These results justify the use of non-optimal Dubins airplane paths for path planning.

Setup - frequency of intermediate altitude case

Alg. 11 yields non-optimal paths for intermediate altitude cases. Therefore it is of interest to know what portion of all start-goal configurations is the intermediate altitude case. We provide experimental results to this question. The experiment was conducted in an empty map. The portion of intermediate altitude cases depends on the extents of the flying area. If the vertical extent of the flying area is very large compared to the horizontal extent, high altitude cases make the largest portion. If the vertical extent of the flying area is very small compared to the horizontal extent, low altitude cases make the largest portion. A realistic ratio for fixed-wing aerial vehicles in outdoor applications is to have a box-shaped flying area with the vertical extent being two-thirds of the horizontal extent and a square horizontal cross section. For each run, the start state was set to $x_I = (0, 0, 0, 0)^T$ and 10000 goal states were randomly sampled in a box with extents in $x\text{-}, y\text{-}$ and $z\text{-}$direction of $l \times l \times \frac{2}{3}l$ for different values of $l$. A realistic value for $l$ is in between 1000 m and 1500 m.
49

4.2. Preliminary results

Results - frequency of intermediate altitude case

The results (fig. 4.5) show that for outdoor missions, the portion of intermediate altitude cases is below 10%. Therefore, the non-optimality of the paths computed with alg. 11 are not of much consequence.

Figure 4.5: The portion of low, intermediate and high altitude cases for different extents of the flying zone $l \times l \times \frac{2}{3} \cdot l$.

4.2.4 Manual code optimizations

Manual code optimization was done in order to further speed up the computation of paths for the Dubins airplane.

Setup

The path planning framework was run 100 times with the optimized code and 100 times with the non-optimized code, both compiled in release mode including compiler optimizations. Every run lasted 0.75 seconds. For planning, D-RRT* was used. The map, start and goal configuration which was used is shown in fig. 4.6.

Results

The results show that due to manual code optimizations (tab. 4.3):

- the median number of iterations in the same timespan for a sampling-based motion planning algorithm increased by 86%. This yielded faster and denser exploration of the space.
- a larger amount of exact paths was found (paths which reached the goal state exactly)
- the median solution length decreased.

Conclusively, manual code optimization improved the overall performance of the path planning framework.
Chapter 4. Experiments and results

Figure 4.6: The map, start state and goal state on which the manual code optimizations were examined.

Table 4.3: An increase of the median number of iterations of 86% within the same planning time was achieved due to manual code optimizations. As a result, more frequently a solution exactly reaching the goal state was found and the median solution length decreased.

<table>
<thead>
<tr>
<th></th>
<th>Non-optimized</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median number of iterations</td>
<td>2800</td>
<td>5200</td>
</tr>
<tr>
<td>Exact solution found in %</td>
<td>95</td>
<td>100</td>
</tr>
<tr>
<td>Median solution length</td>
<td>515</td>
<td>500</td>
</tr>
</tbody>
</table>
4.3 Final results

4.3.1 Block experiment

The block experiment was used to demonstrate the advantages of informed sampling, in particular if the space is large compared to the length of the shortest path between start and goal.

Setup

The block experiment consisted of six different maps (fig. 4.7). In all maps, the obstacle (square pillar) had a side length of 250m and the distance between start and goal was 500m (for details about the map see fig. B.1). From the first map to the sixth map, the ratio \( \frac{l}{d} \) increased from one to six. Every planner was run 100 times on each map. The planning was stopped when the planner found a path whose length is shorter than 105% of the shortest path for the airplane or if the planning lasted for more than 100s without finding a sufficiently short path. For this experiment, the anytime variant of FMT* was used (sec. 3.4.6).

Figure 4.7: Maps, start state and goal state used for the block experiment. Every map contained a square pillar as obstacle surrounded by free space. The whole map had the form of a cube. In all of the six maps (only three are shown in this figure), the pillar had a side length of 250m and the distance between start and goal was \( d = 500m \). In the first map (left) the side length of the space \( l \) surrounding the pillar was the same length as the distance between start and goal. The ratio between the distance between start and goal and the side length of the map was \( \frac{l}{d} = 1 \). In the fourth map (middle) this ratio was \( \frac{l}{d} = 4 \). In the sixth map (right) the ratio was \( \frac{l}{d} = 6 \).

Results

The results show that (fig. 4.8)

- Our implementation produces similar results as described in [17]. This indicates that our implementation is working as expected.

- Exploring the whole space is inefficient when the space is large compared to the distance between start and goal (D-RRT*, D-FMT*).

- Using Dubins adapted sampling (sec. 3.4.3) reduces the time to find a short solution (DA-IRRT*).

- Using a modified euclidean distance (eq. 3.4) to search an initial solution further reduces the time to find a short path (CA-IRRT*). Note that the range parameter for CA-IRRT* was smaller than for D-RRT*, D-IRRT* and DA-IRRT*. This allows CA-IRRT* to find an initial solution similarly fast as D-RRT*, D-IRRT* and DA-IRRT* (fig. B.3), and the subsequent focused
planning is done with a denser tree of motions. This corresponds to a denser exploration of the space and favors the planning of close to optimal paths.

- D-BIT*, DA-BIT* and CA-BIT* all use a heuristic to favor the exploration of the space along the direct connection of start and goal. Hence, Dubins adapted sampling does not improve the convergence rate to find a better solution a lot.

For results concerning the time to find an initial path, the corresponding lengths and the behavior of the convergence, read sec. [B.1](#).

**Figure 4.8**: Block experiment: The median computation time needed by the planners to find a path within 5% of the optimal length for the Dubins airplane for various ratios $\frac{l}{d}$. Error bars denote the nonparametric 95% confidence interval for the median number of iterations calculated from 100 independent runs. [17](#)

### 4.3.2 Gap experiment

The gap experiment was used to show that focused search favors finding difficult passages in a map such as a small gap, in particular if the gap is narrow.

**Setup**

The gap experiment consisted of six different maps (fig. [4.9](#)). In all maps, the obstacle was a rectangular pillar with a gap of different widths for each map (for details see fig. [B.5](#)). Every planner was run 100 times on each map. The planning was stopped when the planner found a path through the gap with adequate length. For this experiment, the anytime variant of FMT* was used (sec. [3.4.6](#)).

**Results**

The results show that (fig. [4.10](#))
Our implementation produces similar results as described in [17]. This indicates that our implementation is working as expected.

Informed sampling decreases the time to find a path through the gap. It increases the density of sampled points in the region around the shortest path. Hence, also the probability of sampling a point in a difficult passage increases, if the passage is part of the shortest path.

Planners using batches of samples work better to find paths through the gap, since when using batches of samples it is likely for many samples to have a nearest neighbor which is connectable with a collision-free dynamically feasible path also in narrow passages (sec. 4.4).

Using Dubins adapted sampling (sec. 3.4.3) reduces the time (DA-BIT*) or at least did not increase the time (DA-IRRT*) to find a path through the gap, especially if the passage is narrow.

Using a modified euclidean distance (eq. 3.4) to search an initial solution reduces the time to find a short path (CA-IRRT* and CA-BIT*). Note that the range parameter for CA-IRRT* was smaller than for D-RRT*, D-IRRT* and DA-IRRT* (sec. 4.1). As a consequence, the tree of motions grows slower and more dense, which is to the advantage of finding a path through a difficult passage.

4.3.3 Random map experiment

The random map experiment was used to evaluate the different planners on an arbitrarily cluttered map.

Setup

Each planner was run 100 times for 10 seconds on the map shown in fig. 4.11. For this experiment, the variant of FMT* predicting the amount of samples workable in the time remaining for planning was used (sec. 3.4.6). The time used as reference for predicting the amount of samples was set to 5 seconds for FMT*.

Results

The results show that (fig. 4.12 and 4.13)
Chapter 4. Experiments and results

Figure 4.10: Gap experiment: The median computation time needed by the different planners to find a path cheaper than flanking the obstacle with adequate length for various gap ratios, $\frac{h_g}{h}$ for the map defined in fig. B.5. Error bars denote a non-parametric 95% confidence interval for the median number of iterations calculated from 100 independent runs. [17]

Figure 4.11: The random map, start and goal state used for the experiment. The map is a cube of 2000m side length with pillars as obstacles at random coordinates.
• Using Dubins adapted sampling (sec. 3.4.3) reduces the time to converge fast to a good quality solution (DA-IRRT* and DA-BIT*).

• Using a modified euclidean distance (eq. 3.4) to search an initial solution reduces the time to find a short path (CA-BIT*). The reason for the long paths resulting from planning with CA-IRRT* is the following: Initial paths found with CA-IRRT* are long. This is due to the fact, that the modified euclidean distance used for searching an initial path (eq. 3.4), underestimates the true length of a Dubins airplane path. Additionally, the range parameter of CA-IRRT* was smaller than for D-RRT*, D-IRRT* and DA-IRRT* which makes the tree of motions grow slower and more dense (tab. 4.2). Therefore, the tree of motions contains many states already for solutions with low quality (long paths). In this case, it is computationally expensive to add new states because a lot of nearest neighbor computations have to be processed. Conclusively, the convergence to the optimal path is delayed.

• The typical behavior of our two-stage D-FMT* planning approach is visible (sec. 3.4.6): In more than 95% of the runs, D-FMT* found an initial solution. A new batch of samples was drawn (the amount was computed according to the time needed to process the first batch of samples and the complexity of the algorithm). After the processing of the second batch of samples (approximately 3 seconds), FMT* finds the final solution. As described in sec. 3.4.6 the amount of samples predicted by D-FMT* underestimates the true amount of samples that would be workable in 5 seconds.

• A comment on planners using batches of samples and running a graph search algorithm to find the shortest path: The fact that the earliest solutions are short paths is due to the behavior of graph search algorithms (e.g. in D-FMT*). If the network resulting from the samples allows for a short path which is found early, a lot of samples do not have to be considered, hence the batch of samples is processed fast. The shorter the path is, the more samples can be skipped and the earlier the algorithm finishes the first batch of samples and returns the solution.

4.3.4 Slalom experiment

The slalom experiment aimed at showing that planners using batches of samples (D-FMT*, D-BIT*, DA-BIT*, CA-BIT*) have advantages in such maps. Planners iteratively sampling one sample at a time and attempting to connect it to the existing tree of motions have difficulties finding a path.

Setup

The slalom experiment was conducted in the map shown in fig. 4.14. Each planner was run 100 times for 20 seconds. For this experiment, the variant of FMT* predicting the amount of samples workable in the time remaining for planning was used (sec. 3.4.6). The time used as reference for predicting the amount of samples was set to 10 seconds for FMT*.

Results

The results show that (fig. 4.15)

• In more than 95% of the runs, D-RRT*, D-IRRT* and DA-IRRT* did not find a solution within 20 seconds in the slalom map. All of these planners grow
Figure 4.12: Random map experiment: Median length of the current best path over time for different planners. The same results including the 5th and the 95th percentile are shown in fig. B.6. Note that the lengths of the paths found with D-RRT* and D-FMT* are too long to be seen in this plot. They can be seen in fig. B.6.

Figure 4.13: Random map experiment: The time needed to find an initial solution and the lengths of the initial solutions. The median is plotted in red, the 25th and 75th percentiles are plotted horizontally in blue and the 5th and 95th percentiles are plotted horizontally in black.
4.3. Final results

Figure 4.14: The map, start state and goal state used for the slalom experiment.

a tree of motions by iteratively sampling one sample at a time and trying to connect it to the existing tree. This tree has to grow through the whole map. At the beginning, when the tree of motions is still close to the start state, if the randomly sampled state is separated from the tree of motions by at least one wall, it is unlikely that there exists a collision-free flyable path to connect the sampled state to the existing tree of motions. Hence the tree of motions grows slowly. This is why in many runs there was still no solution after 20 seconds in slalom map (fig. 4.16 and 4.24).

- D-FMT*, D-BIT*, DA-BIT*, CA-BIT* all found a solution in every run and found an initial solution fast (fig. 4.25).

- Using a modified euclidean distance (eq. 3.4) to search an initial solution reduces the time to find an initial solution whereas the quality of the solution is worse (CA-IRRT*, CA-BIT*). Convergence rate is fast at the beginning such that after 1 second, the quality of the solution is similar to paths found using only Dubins paths.

- The typical behavior of our two-stage D-FMT* planning approach is visible as already described in the results of the random map experiment (sec. 4.3.3).

- Note that informed sampling (sec. 3.4.2) has no influence on the time necessary to find an initial solution.

- Note that in the slalom map (fig. 4.14), Dubins adapted sampling could not improve the convergence rate of the planners since the vertical extension of the map was small and sampling an ellipsoid resulted in sampling a slice of the ellipsoid due to the fact that samples outside of the map were rejected.

- The behavior of FMT* is visible. Find an initial solution with a low number of samples, predict the amount of samples workable in 10 seconds and replan (sec. 3.4.6).

For results concerning the time to find an initial path and the corresponding lengths, read sec. B.4.

4.3.5 Mountain experiment

The mountain experiment was used to examine the performance of the planners in real maps and environments that are encountered in typical fixed-wing outdoor
Chapter 4. Experiments and results

Figure 4.15: Slalom experiment: Median length of the current best path over time for different planners. In more than 95% of all runs, D-RRT*, D-IRRT* and DA-IRRT* did not find a solution (not plotted). The same results including the 5th and the 95th percentile are shown in fig. B.8.

Figure 4.16: Tree of motions (slowly) growing through the map (DA-IRRT*). The motions are shown in black. The current best solution (closest to the goal state) is shown in red. The last figure shows the tree of motions after graph pruning, this is why it is less dense, and the final solution after 40 seconds.
4.3. Final results

(a) CA-IRRT*: Visualization of different solutions at different times.

(b) CA-BIT*: Visualization of the initial batch of samples and the initial solution (black) and the final solution found after 20s (red).

Figure 4.17: Visualization of results.

missions. The chosen map represented a realistic field of application with a mountain where the airplane had to find a path from one valley to the other valley over a mountain pass.

Setup

The mountain experiment was conducted in the map shown in fig. 4.18. Each planner was run 100 times for 10 seconds. For this experiment, the variant of FMT* predicting the amount of samples workable in the time remaining for planning was used (sec. 3.4.6). The time used as reference for predicting the amount of samples was set to 5 seconds for FMT*.

Figure 4.18: The map, start state and goal state used for the Mountain experiment.
Chapter 4. Experiments and results

Results

The results show that (fig. 4.19 and 4.20)

- Dubins adapted sampling reduces time to converge to short paths.

- Using a modified euclidean distance (eq. 3.4) to search an initial solution (CA-IRRT*, CA-BIT*) reduces the time to find an initial solution whereas the quality of the solution is worse. Convergence rate for CA-BIT* is fast at the beginning such that the after 1 second, the quality of the solution is similar to paths found using only Dubins paths and after 1 second, the paths found by CA-BIT* are shorter as the paths found by D-FMT*, D-BIT* and DA-BIT*.

- The median time required to find an initial solution for D-RRT*, D-IRRT* and DA-IRRT* was more than a second in the mountain experiment (fig. 4.20).

- CA-IRRT* did not find short paths in this experiment, however, in less than 5% of the runs the time to find an initial solution for CA-IRRT* exceeded a second. The reason for the long paths resulting from planning with CA-IRRT* was the same as for the random map experiment (sec. 4.3.3).

- The typical behavior of our two-stage D-FMT* planning approach is visible as already described in the results of the random map experiment (sec. 4.3.3).

- The behavior of D-FMT*, D-BIT* and DA-BIT* at the beginning of planning is due to the underlying graph search algorithm (sec. 4.3.3 contains a more detailed description on this behavior).

![Median path lengths over time](image)

Figure 4.19: Mountain experiment: Median length of the current best path over time for different planners. The same results including the 5th and the 95th percentile are shown in fig. B.11.
4.3. Final results

Figure 4.20: Mountain experiment: The time needed to find an initial path and the lengths of the initial paths. The median is plotted in red, the 25th and 75th percentiles are plotted horizontally in blue and the 5th and 95th percentiles are plotted horizontally in black. In this experiment, in more than 95% of the runs D-RRT*, D-IRRT* and DA-IRRT* did not find an initial path within 1 second. Conclusively, for this experimental setup, they do not fulfill the real-time requirements asked for (sec. 3). For CA-BIT*, the time to plan the initial path is not visible in the plot because it is too short. In some runs, FMT* did not find an initial path with the initial batch of samples, but only in the subsequent batches when FMT* searched for an initial path with twice as much samples as in the initial batch of samples (alg. 12). Therefore, some values for the time to find an initial path and the corresponding lengths are 0, as these values were initialized with 0. Note that the median and the other plotted statistical values are not influenced by these outliers.

4.3.6 Kandervalley experiment

The Kandervalley experiment was used to examine the performance of the planners in large real maps (path lengths up to 15km) and their flexibility regarding the size scale of the map. The experiment consisted of two setups in the same map. The first setup allowed the airplane to fly everywhere in the map. The second setup restricted the airplane to stay below 750m (relative to the lowest point on the map).

Setup

The Kandervalley experiment was conducted in the map shown in fig. 4.21. Each planner was run 100 times for 10 seconds. For this experiment, the variant of FMT* predicting the amount of samples workable in the time remaining for planning was used (sec. 3.4.6). The time used as reference for predicting the amount of samples was set to 5 seconds for FMT*.

Results

The results showed that (fig. 4.22)

- The planners find an initial solution fast (in at least 95% of the runs an initial path was found in less than 1 second, except for D-FMT*) and plan short paths in adequate time (at about 2 seconds) in a large real map (fig. B.14).

- Dubins adapted sampling improves convergence rate to a short solution (DA-IRRT* and DA-BIT*).

- The reason for the long paths resulting from planning with CA-IRRT* is the same as for the random map experiment (sec. 4.3.3). Note that the range
The parameter for CA-IRRT* was the same than for D-RRT*, D-IRRT* and DA-IRRT*. Still, the tree of motions contains many states already for solutions with low quality (long paths).

- The typical behavior of our two-stage D-FMT* planning approach is visible as already described in the results of the random map experiment (sec. 4.3.3).

- The behavior of D-FMT* and D-BIT* at the beginning of planning is due to the underlying graph search algorithm (sec. 4.3.3 contains a more detailed description on this behavior).

For results concerning the time to find an initial path and the corresponding lengths, read sec. B.6.

![Figure 4.21](image)

**Figure 4.21:** The map, start state and goal state used for the Kandervalley experiment (left) and the Kandervalley experiment where the airplane was restricted to stay below 750m (right).

![Figure 4.22](image)

**Figure 4.22:** Kandervalley experiment: Median length of the current best path over time for different planners. The same results including the 5th and the 95th percentile are shown in fig. B.13.
Setup - restricted height

The Kandervalley experiment with restricted height was conducted with the start and goal state shown in fig. 4.21. Each planner was run 100 times for 10 seconds. For this experiment, the variant of FMT* predicting the amount of samples workable in the time remaining for planning was used (sec. 3.4.6). The time available for planning was set to 5 seconds for FMT*.

Results - restricted height

The results show that (fig. 4.23)

- The planners find an initial solution fast (in at least 95% of the runs an initial path was found in less than 1 second, except for D-FMT* and D-BIT*) and plan short paths in adequate time (at about 2 seconds) in a large real map even if the path leads through a long valley (fig. B.17).

- Dubins adapted sampling improves the convergence rate to a short solution (DA-IRRT* and DA-BIT*).

- Using a modified euclidean distance (eq. 3.4) to search an initial solution improves the convergence rate towards the optimal solution (CA-IRRT*, CA-BIT*).

- Informed sampling (sec. 3.4.2) does not influence the time necessary to find initial solutions.

- In this experiment, D-BIT* yielded good quality paths to the disadvantage of finding an initial path later than the other planners (for the parameter tuning of the different planners, it was assumed that finding an initial solution within 1 second in at least 95% of the runs is acceptable).

- The typical behavior of our two-stage D-FMT* planning approach is visible as already described in the results of the random map experiment (sec. 4.3.3).

- The behavior of D-BIT* at the beginning of planning is due to the underlying graph search algorithm (sec. 4.3.3 contains a more detailed description on this behavior).

For results concerning the time to find an initial path and the corresponding lengths, read sec. B.6.

4.4 Summary and analysis of the experiments

The summary (tab. 4.4) outlines

- in experiments where the planners either had to find a difficult and specific or twisted path (gap, slalom) or the space to fly is tight (random map, mountain experiment), planners drawing batches of samples (D-FMT*, D-BIT*, DA-BIT*, CA-BIT*) find shorter solutions in lesser time than planners sampling one sample at a time (D-RRT*, D-IRRT*, DA-IRRT*, CA-IRRT*).

- in experiments with wide open spaces planners sampling one sample at a time find shorter paths in lesser time than planners drawing batches of samples.

Points according to their ranking in tab. 4.4 were assigned to each planner. Planners which did not find an initial solution in an adequate time were set to the last rank. Tab. 4.5 shows the ranking of the planners.
Table 4.4: Summary of the experiments. For the block and gap experiment, the planners are listed according to their performance on each map. No information about the time needed to find an initial solution is presented. For the other experiments, the planners are listed according to the length of the path they found after 2 seconds. Remarks are added when some planners did not find an initial solution early enough. Maps and start-goal configurations for which a difficult path has to be found and maps with wide open spaces are marked in green and red respectively.

<table>
<thead>
<tr>
<th>Block</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random map</td>
<td>Shallow (after 2s)</td>
</tr>
</tbody>
</table>

Remarks

1. CA-IRRT*, DA-IRRT*:
   In more than 5% of the runs, these planners did not find an initial solution within 1 second.
2. D-IRRT*, CA-BIT*, D-BIT*:
   In more than 5% of the runs, these planners did not find an initial solution within 1 second.
3. D-RRT*:
   In more than 5% of the runs, this planner did not find an initial solution within 1 second.
4. D-FMT*, D-BIT*:
   In more than 5% of the runs, these planners did not find an initial solution within 1 second.
4.4. Summary and analysis of the experiments

Figure 4.23: Kandervalley with restricted height experiment: Median length of the current best path over time for different planners. The same results including the 5th and the 95th percentile are shown in fig. B.16.

Table 4.5: Points of planners according to tab. 4.4. The points for each planner are computed as the sum of its ranks over all respective maps. The lower the amount of points the better did the planner perform.

<table>
<thead>
<tr>
<th></th>
<th>D- RRT*</th>
<th>D- IRRT*</th>
<th>DA- IRRT*</th>
<th>CA- IRRT*</th>
<th>D- FMT*</th>
<th>D- BIT*</th>
<th>DA- BIT*</th>
<th>CA- BIT*</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>45</td>
<td>26</td>
<td>23</td>
<td>21</td>
<td>48</td>
<td>25</td>
<td>19</td>
<td>17</td>
</tr>
<tr>
<td>Green</td>
<td>25</td>
<td>18</td>
<td>18</td>
<td>16</td>
<td>25</td>
<td>10</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Red</td>
<td>20</td>
<td>8</td>
<td>5</td>
<td>4</td>
<td>23</td>
<td>15</td>
<td>13</td>
<td>12</td>
</tr>
</tbody>
</table>
Minimum path length

Wide open spaces  Pursuant to the ranking, CA-IRRT* is the best choice for planning in wide open spaces. In the same environments, DA-IRRT* yields similarly well paths. Both planners share commonalities, but have also advantages and disadvantages different from each other. As soon as a path between start and goal is known, CA-IRRT* and DA-IRRT* focus the search on a region adapted for the Dubins airplane $X_{\bar{c}, D}$ (sec. 3.4.3). This leads to a dense exploration of the region which potentially can improve the current best path. CA-IRRT* benefits from finding an initial path early whereas DA-IRRT* needs more time to find an initial path than CA-IRRT*. The initial path found when planning with CA-IRRT* is usually longer than the initial path found when planning with DA-IRRT*. In many cases, the difference in time for finding an initial path between DA-IRRT* and CA-IRRT* is negligible (block experiment, sec. 4.3.1, and the two Kandervalley experiments, sec. 4.3.6). In the mountain experiment (tight air space), in more than 5% of the runs, DA-IRRT* did not find an initial path in less than a second. The setup for the mountain experiment is similar to a real application, hence real-time requirements should be fulfilled, which is not the case for DA-IRRT*.

Often, after two seconds, the paths found with CA-IRRT* are as short or shorter than the paths found by DA-IRRT*. This is due to the fact, that CA-IRRT* finds an initial path fast and therefore can focus its search earlier than DA-IRRT*, which is more efficient than exploring the whole space.

When the initial path found with CA-IRRT* is long compared to the initial path found with DA-IRRT*, CA-IRRT* yields longer paths than DA-IRRT* also after two seconds. This happens for example if the path leads over a mountain pass or any other elevation (mountain experiment, sec. 4.3.5, and the Kandervalley experiment sec. 4.3.6), or if the map is cluttered with a lot of obstacles (random map experiment, sec. 4.3.3). DA-IRRT* always computes with the true lengths of the Dubins airplane paths. Therefore the initial paths found with DA-IRRT* are shorter than initial paths found with CA-IRRT*, in particular if the airplane has to cover a lot of meters in altitude.

If an initial path is required rapidly and a loss in the quality of the initial path (a longer path) is acceptable, CA-IRRT* is the planner to choose. If a short initial path is required and longer times to find it are acceptable, DA-IRRT* is the planner to choose.

Cluttered and difficult maps  When the map is cluttered with obstacles, the path has narrow passages which are difficult to find or it is twisted, CA-BIT* finds the shortest path in the least amount of time. Planning with CA-BIT* always yields path which are as short or shorter than the paths resulting from D-FMT*, D-BIT* or DA-BIT*, in lesser time.

CA-BIT* is the planner to choose.

Summary of CA-IRRT* and CA-BIT*, the best planners for planning in wide open spaces and cluttered difficult maps respectively  CA-IRRT* and CA-BIT* benefit from finding an initial solution early and even though the initial path is may be long, after two seconds, the paths are as short or shorter than the paths found by other planners. As soon as a path between start and goal is known, CA-IRRT* and CA-BIT* focus their search on a region adapted for the Dubins airplane $X_{\bar{c}, D}$ and behave like DA-
IRRT* and DA-BIT* respectively. This leads to a dense exploration of the region which potentially can improve the current best path $X_{c,D} \subset X_{\hat{c},D}$.

When the map contains wide open spaces, CA-IRRT* finds the shortest path in the least amount of time. When the map is cluttered with obstacles, the path has narrow passages which are difficult to find or it is twisted, CA-BIT* finds the shortest path in the least amount of time.

CA-IRRT* samples one sample at a time and attempts to connect it to the existing tree of motions with a Dubins airplane path. Furthermore, the maximum length of a motion is limited according to the range parameter (sec. 4.1). This can be iterated fast. Therefore, in wide open spaces, the tree of motions grows fast (compare the plots for graph motions in sec. B.1). Due to the range parameter, the tree densely explores the space. This allows to find short paths to the goal rapidly. CA-BIT* draws batches of samples all at once. These are distributed all over the space (or all over the region where the search is focused to $(X_{\hat{c},D})$ once a path is known). Many of the samples are not used and hence not added to the tree of motions. Hence, the space is explored less densely resulting in lower quality paths (compare the plots for graph motions in sec. B.1).

In cluttered maps or if the path has narrow passages or is twisted, sampling one state and attempt to connect it to the tree of motions fails in many cases since it is likely that there is an obstacle between the sample and the tree of motions blocking the way (the non-optimal Dubins airplane path connecting the tree of motions to the sample is not collision-free). Hence, many samples are rejected. This makes the tree grow slowly (fig. 4.24). In cluttered maps or if the path has narrow passages or is twisted, drawing batches of samples has the advantage that many samples have a nearest neighbor for which a collision-free path between them exists. Hence, it is likely to find an initial path already with the first batch of samples, even in difficult environments (fig. 4.25).

Figure 4.24: The sampling and planning process of CA-IRRT* one step at a time. CA-IRRT* does not work efficiently in maps cluttered with obstacles or if the path has difficult passages or it is twisted.

Arbitrary optimization objective

The long-term goal for many fixed-wing aerial vehicles is to plan and fly minimum-energy paths. In order to do so, motion planning algorithms able to handle arbitrary cost functions have to be used. Hence, for planning in wide open spaces for arbitrary optimization objectives, D-RRT* yielded good results.
Figure 4.25: The sampling and planning process of CA-BIT* one step at a time. CA-BIT* does work efficiently in maps cluttered with obstacles or if the path has difficult passages or it is twisted.

When the map is cluttered with obstacles, the path has narrow passages which are difficult to find or it is twisted and the optimization objective is arbitrary, D-FMT* yielded good results.
Chapter 5

Conclusion and future work

5.1 Conclusion

From sec. 4.4, the following conclusion ensues.

- Planning for fixed-wing aerial vehicles in wide open spaces (e.g. mountainous regions, above large cities with skyscrapers, above forests with large trees) and searching for shortest paths:
  - CA-IRRT* has yielded the shortest path in the least amount of time. CA-IRRT* also fulfills all requirements asked for (sec. 3).
  - If the path comprises a lot of meters in altitude relative to the horizontal distance, DA-IRRT* has yielded shorter paths than CA-IRRT*. DA-IRRT* also fulfills all requirements asked for (sec. 3).

- Planning for fixed-wing aerial vehicles in maps cluttered with obstacles or if the paths are twisted or lead through narrow passages (e.g. indoors, in cities between houses, in forests) and searching for shortest paths:
  - CA-BIT* has yielded the shortest path in the least amount of time. CA-BIT* also fulfills all requirements asked for (sec. 3).

- Planning for fixed-wing aerial vehicles in arbitrary maps and search for shortest paths:
  - CA-BIT* has consistently yielded short paths in a reasonable amount of time. CA-BIT* also fulfills all requirements asked for (sec. 3).

If the optimization objective is different from path length, it is not possible to use informed planners in the way they are presented in this thesis. Hence, D-RRT*, D-FMT* or any other control-based path planning algorithm which is able to handle arbitrary cost functionals of the form eq. 2.1 should be used.

- Planning for fixed-wing aerial vehicles in wide open spaces (e.g. mountainous regions, above large cities with skyscrapers, above forests with large trees) and optimizing an arbitrary optimization objective:
  - D-RRT* has yielded the shortest path in the least amount of time.

- Planning for fixed-wing aerial vehicles in maps cluttered with obstacles or if the paths are twisted or lead through narrow passages (e.g. indoors, in cities between houses, in forests) and optimizing an arbitrary optimization objective:
  - D-FMT* has yielded the shortest path in the least amount of time.
This thesis considers planning shortest paths for fixed-wing aerial vehicles in wide open spaces. For this scenario, CA-IRRT* is most practical.

Figure 5.1: A collision-free shortest path for a fixed-wing aerial vehicle found with CA-IRRT* around Eiger, Mönch and Jungfrau (left to right), Switzerland.
5.2 Future work

Meteorological influences

The long-term goal for many fixed-wing aerial vehicles is to plan minimum energy paths. Wind and rain strongly affect the energy consumption of the airplane when following a trajectory. Hence, they should be considered to find paths resulting in less energy consumption and possibly less flight time. For solar aerial vehicles, cloud coverage may be considered to find paths with enough insolation on the solar panels to achieve longer flight times.

There are three possibilities:

- Minimum flight time paths: For finding minimum flight time paths (in case of wind, time-optimal paths are not necessarily the same as minimum length paths), wind (and possibly other meteorological influences) should be considered. This will probably yield paths with lower energy consumption when tracking the path. McGee, Spry and Hedrick [25] and Bakolas and Tsiotras [26] present tools to compute time-optimal paths in the case of constant wind.

- Minimum energy paths: Instead of searching for time-optimal paths, energy consumption can be minimized directly. Planning minimum energy paths involves the modeling of the meteorological influences with the strongest impact on the airplane, the modeling of the airplane such that energy consumption is realistically represented and defining a reasonable cost functional thereof.

Informed sampling for the Dubins airplane for arbitrary cost functionals

Informed sampling, as presented and used in this thesis only works for minimizing path length. However, the idea of only sampling states which can improve current best path, assuming obstacle free space, is theoretically applicable to arbitrary cost functionals. Let $c(x)$ be the cost of an optimal path from $x_I$ to $x_G$ constrained to go through $x$. The informed subset has the form of eq. 2.2.

In the case of a system without differential constraints and the cost function being path length, $X_c$ turns out to be an ellipsoid. Samples can be drawn directly from an ellipsoid (e.g. no rejection sampling required). This makes it particularly efficient as the path approaches the optimum.

The informed subset for the Dubins airplane $X_{c,D}$ and minimizing path length has a more difficult shape and cannot directly be sampled. However, the informed subset of the Dubins airplane for minimum path length is contained in the informed subset for a system without differential constraints $X_{c,D} \subset X_c$. Therefore, rejection sampling on samples drawn from $X_c$ can be applied (e.g. as presented in sec. 3.4.3). The informed subset for the Dubins airplane and an arbitrary cost functional has a complex, and in general unknown shape and in general cannot be sampled directly. Therefore, when minimizing arbitrary cost functionals, other tools than directly sampling the informed subset have to be applied.

- Hierarchical rejection sampling as presented by Kunz, Thomaz and Christensen [36].

- Define a probability distribution all over the space according to the cost functional and draw samples thereof.

- Allow each sample to sway in a local neighborhood to move the samples towards low cost areas or split the sample into several samples distributed in the local neighborhood.
In the specific case of a slow-flying solar-powered UAV, sampling could be focused on low wind speed regions or winds pointing in direction of the goal or regions with above-average sun irradiation.

**Informed sampling for the Dubins airplane for path length**

As described above, the informed subset for minimum path length for the Dubins airplane $X_{c,D}$ is contained in the informed subset for a system without differential constraints $X_c$, which is an ellipsoid. In this thesis, it was approximated by cutting off the top and the bottom of the ellipsoid. More exact approximations could be done by further adopting the value of the semi-principal axis of $X_c$ to the Dubins airplane paths (fig. 5.2).

![Diagram](image)

**Figure 5.2:** The informed subset for the Dubins airplane for minimum path length can be approximated by adjusting the value of the semi-principal axis. Consider path length as optimization objective. The informed subset for a system without differential constraints is drawn as solid line in black. An approximation to the informed subset for the Dubins airplane is drawn dashed in black.

**Non-uniform sampling of the informed subset**

Currently, the informed subset (in our case an ellipsoid, possibly with the top and the bottom cut off) is sampled uniformly. This leads to a lot of samples being in the outer region of the ellipsoid since there is most of the volume.

- Similar arguments as presented for BIT*, which explores the space by expanding from the vertex with the lowest cost-to-go value, could be used to justify sampling procedures which focus the search more on regions close to the straight-line connection between start and goal.
- Another idea would be to use probability distributions similar to the Gaussian distribution (maybe skewed towards the goal) to sample around the vertex with the lowest cost-to-go value.

**Determine yaw in a smart way**

Currently, the yaw of the Dubins airplane motion model is randomly sampled. This is disadvantageous, especially in narrow corridors. Hence, in some situations,
determining the yaw depending on the map, the cost functional or heuristics may yield advantages.

**Limit yaw using information from an early path**

An idea was, to first search for a path as usual (randomly sample the yaw). After some time or as soon as a reasonably good path has been found, the yaw of the optimal path at each point is assumed to be within a range of the yaw of the current best path at the state perpendicular to the straight line connecting start and goal (fig. 5.3). For points in the space where no path exists all over the space perpendicular to the line connecting start and goal, the yaw could be randomly sampled or be assumed to point towards the goal. Helices for the Dubins airplane have to be considered as special cases.

![Figure 5.3: Restricting the yaw for each point in the space according to an existing path (upper path).](image)

**Free yaw condition at the terminal state**

Thomaschewski [37] presented the solution for Dubins paths where the terminal direction is not prescribed.

Hota and Ghose [38] presented the solution for Dubins paths where the terminal straight line is specified but not the final position.

With these results, it is possible to determine the position or yaw, instead of randomly sampling it. Conclusively, only the position has to be sampled whereas the yaw is arbitrary. Once, the path was computed, the position and yaw of the newly added state is fixed. The expansion of the search graph is then continued knowing the position and orientation at each vertex.

**Extensions for real-time planning**

In order to make the current path planning framework real-time capable, some features have to be added.

- Make the paths dynamically feasible (sec. 5.2), e.g. by applying polynomial path smoothing or think about an adequate minimum turning radius such that the airplane can reasonably track paths with non-continuous curvature.
• Currently, when reloading a new map, the framework is restarted. This can be improved by only reloading the map and the variables required to be reinitialized.

• When planning with a height map, the octomap is loaded for visualization. This requires some time (1-2 seconds for large maps). This is not necessary for real applications and should be switched off.

• Make sure not to enter the region of inevitable collision ($X_{ric}$). These are states from which entry into $X_{obs}$ will eventually occur (for more details consider [9], sec. 14.1.3.2). This includes estimating the time needed for planning, the distance to the next obstacle, being aware about the speed relative to the ground, the minimum turning radius, and then commit the command to a fast obstacle avoidance algorithm early enough. Note that finding a path may take longer as the airplane gets closer to an obstacle since it gets more difficult to find a path out of the tight region.

Dynamically feasible paths

Dubins paths are not really dynamically feasible due to the non-continuous curvature at the junction of path segments. There are ways to handle this.

• Dubins paths with continuous curvature: Scheuer and Laugier [27] and Skari et al. [28] present methods to generate continuous curvature paths. These could be used instead of the Dubins airplane paths presented by Chitsaz and Lavalle[16].

• Polynomial Trajectory Planning: Polynomial trajectory planning generates paths with continuous curvature as well. Richter, Bry and Roy [29] presented continuous curvature paths for fixed-wing aerial vehicles based on Dubins curves.

Plan on a 2D plane through start and goal

Planning in two dimensions is faster than planning in three dimensions. An approach to rapidly find paths is to define a plane which goes through the start and the goal state. Additionally the inclination of the plane has to be specified. Subsequently, two-dimensional path planning can be done, e.g. with Dubins car paths. Note that due to the restriction to stay on the plane, there is no guarantee to find globally optimal paths.

Optimal Dubins airplane paths

Currently, the intermediate case for Dubins airplane paths is solved suboptimally with alg. 11 for the sake of computational speed. In order to have a guarantee on computing shortest paths, shortest paths for the intermediate case have to be computed. This may require case distinctions or/and optimization procedures. Currently, there exists an implementation for computing optimal Dubins airplane paths. However, not all cases are solved optimally. The implementation is based on the work presented by Beard and McLain [31].

Dubins set classification for short path cases

Currently, the Dubins set classification scheme is only implemented for long path cases [4]. Extending this also for short path cases will speed up the planning.
For most outdoor applications, short path cases are rare. Furthermore, for short path cases, the classification scheme is more complex and less efficient than for long path cases. Therefore, Dubins set classification for sort path may not bring large speed up.

Adaption of planner parameters during planning

A remarkable amount of time during planning is spent on processing samples which are not part of the optimal path and will never be part of the shortest path found by the path planning algorithm. Approaches to improve this were presented in form of focused search on the informed subset and heuristics during graph search algorithms (e.g. for all BIT* variants). Further improvements could be achieved by adapting the planner parameters during the search.

1. To search an initial path, exploration of the whole space is necessary. However, preferably a low amount of samples should be processed, since it is likely that many samples fall in a part of the space which is not relevant for the further planning. Hence, a large “range” parameter for all RRT* variants or a small “samples per batch” parameter for all BIT* variants may be useful.

2. When an initial path is known, it is more relevant to find short paths than quickly find new paths. Therefore, the tree of motions should become denser. Hence, a smaller “range” parameter for all RRT* variants or a larger “samples per batch” parameter for all BIT* variants may be useful.

3. The closer the current shortest path gets to the optimal path, the denser the tree of motions should get in order to find improved paths in reasonable time. Hence, a small “range” parameter for all RRT* variants or a large “samples per batch” parameter for all BIT* variants may be useful.

Note that informed sampling is doing exactly the same (increase the density of samples in the region of interest). However, since it is not possible to exactly sample the informed subset for the Dubins airplane, this could further improve the convergence to close-to-optimal paths. Actually, this yields to the exploration of the informed subset for the Dubins airplane without directly sampling it. Since some parameters strongly affect the performance of some planners, choosing parameters dynamically in a smart way can speed up planning and reduce uncertainty arising due to manual parameter choices.

Note that this can easily be achieved for CA-IRRT* and CA-BIT*. In this thesis, for CA-IRRT* and CA-BIT* the parameters for searching an initial solution and for the subsequent planning were chosen the same. It would be interesting to try different parameters instead.

These arguments only hold for minimizing path length.

Planning with modified euclidean distance only

Instead of planning only the initial path using a modified euclidean distance (eq. 3.4) for distance queries, the full planning procedure could be based on the mentioned distance function. There is no guarantee on converging to the globally shortest path. However, reasonable paths should be found faster than with the planning procedures used in this thesis.

Optimized collision checking

Discrete collision checking discretely checks states along a path to assure the path is collision-free. Important for collision checking are the grid of the map, the true
extent of the aerial vehicle, the extent of the box used for collision checking and the collision checking resolution (corresponds to the maximum distance between two states on a path which are checked for collision). If the extent of the box is the same as the grid resolution then collision checking can done efficiently by checking only at most four values for each collision check.

The smallest reasonable dimensions of the box are the same as the extent of the aerial vehicle. This leads to paths for which there exist relevant regions which are not checked for collision (fig. 5.4) unless the collision checking resolution is chosen large. Choosing the collision checking resolution small leads to checking a lot of regions redundantly. Instead of increasing the collision checking resolution, the box extents can be increased. This leads to checking a lot of regions redundantly and a lot of regions needlessly. Furthermore to make sure at most four values have to be checked for collision, the grid resolution of the map has to be adjusted.

The task is to find the dimensions of the box and a collision checking resolution such that any path including space on its left and right according to the true extents of the aerial vehicle is completely checked for collision.

For an airplane with a wing-span of 10 m and a minimum turning radius of 25 m, choosing a square box with a side-length of 30 m and a collision checking resolution of $\frac{30}{2} m = 15 m$ solves the task (fig. 5.5). Further speed-up in planning can be achieved by solving the described task and minimizing the regions which are checked redundantly and the regions which are checked needlessly.

(a) Collision checking along a path aligned with the map grid (red). Collision checking works as expected.

(b) Collision checking along a straight-line path with an angle of 45° relative to the map grid. There is relevant area which is not checked for collision. This may cause a crash of the aerial vehicle.

(c) Collision checking along a circular path (red). There is relevant area which is not checked for collision. This may cause a crash of the aerial vehicle.

Figure 5.4: Discrete collision checking along a circular path (red) with a non-rotating square box (black). The maximum extent of the aerial vehicle is dashed in blue (10 m). The dimension of the square box is the same as the maximum extent of the aerial vehicle. The collision checking resolution is the same as the dimension of the square box.
Future work

(a) Collision checking along a straight-line path aligned with the map grid (red).
(b) Collision checking along a straight-line path with an angle of 45° relative to the map grid (red).
(c) Collision checking along a circular path (red). There is relevant area which is not checked for collision. This may cause a crash of the aerial vehicle.

Figure 5.5: Discrete collision checking along a path (red) with a non-rotating square box (black). The maximum extent of the aerial vehicle is dashed in blue (10 m). The dimension of the square box is three times the maximum extent of the aerial vehicle. The collision checking resolution is half of the dimension of the square box. For arbitrary Dubins airplane paths with $r_{\text{min}} = 25$ m, all relevant area is checked for collision. However, there are regions which are checked redundantly and regions which are checked needlessly.
Figure 5.6: A path with a length of 6010 m planned with CA-IRRT* around Finsteraarhorn, Switzerland. The start and goal are colored in red and green respectively. Planning was interrupted after 2 s. An initial path with a length of 10419 m was found after 0.003 s.
Bibliography


Appendix A

Additional material

A.1 Proof: Eq. 3.6 is a pseudometric

- The euclidean distance $d_{eucl}$ is a metric on $\mathbb{R}^3$.
- The maximum of two metrics is again a metric [39].
- The distance approximating the distance of Dubins paths $d_{DubinsApprox}(x_1, x_2) = \frac{|x_1.z - x_2.z|}{\sin \gamma_{max}}$ with $\gamma_{max} \in (0, \pi)$ is a pseudometric on $\mathbb{R}^3$.

Let $x_1, x_2, x_3 \in \mathbb{R}^3$

\begin{align*}
- d_{DubinsApprox}(x_1, x_2) & \geq 0 \\
- d_{DubinsApprox}(x_1, x_2) &= \frac{|x_1.z - x_2.z|}{\sin \gamma_{max}} = \frac{|x_2.z - x_1.z|}{\sin \gamma_{max}} = d_{DubinsApprox}(x_2, x_1) \\
- d_{DubinsApprox}(x_1, x_2) &= \frac{|x_1.z - x_2.z + x_3.z - x_3.z|}{\sin \gamma_{max}} \leq \frac{|x_1.z - x_3.z|}{\sin \gamma_{max}} + \frac{|x_3.z - x_2.z|}{\sin \gamma_{max}} = d_{DubinsApprox}(x_3, x_2)
\end{align*}

A.2 Proof: The length of non-optimal Dubins airplane paths is a premetric

The length of optimal Dubins airplane paths is a quasimetric. Fig. [A.1] shows that for the length of non-optimal Dubins airplane paths computed with alg. 9, 10 and 11 the triangle inequality is not fulfilled. Therefore, the length of the non-optimal Dubins airplane paths is not a quasimetric. Since the length of the paths is nonnegative and zero if and only if the start and the goal is the same, the length of the non-optimal Dubins airplane paths is a premetric.
Figure A.1: Counterexample to prove that the triangular inequality is not satisfied when the distance function is defined as the length of the non-optimal Dubins airplane paths computed from algs. 9, 10, 11. Assume the start goal configuration (blue arrows) are intermediate altitude case and there exists a path with maximum climb angle. Assume, the goal is slightly above the height where the airplane would fly when flying a 180° helix with $r = r_{\text{min}}$ and maximum climb angle $\gamma_{\text{max}}$. Using alg. 11 the red path results. However, a shorter path is the green path going through configuration i.
Appendix B

More results

All results for all experiments are available in the “Results” folder supplied with the thesis.

B.1 Block experiment

Setup

The block experiment consists of six maps. Fig. B.1 shows details about the dimension of the maps used for the block experiment.

![Diagram of the setup on the second map of the block experiment.](image)

Figure B.1: Scheme of the setup on the second map of the block experiment. Sectional view at half of the total height (for the second map this is 500m).

Results

The results shown in fig. 4.8 do not show any information about the time to find an initial path, the corresponding lengths and the behavior of the convergence. These results are shown here for the third map (fig. 4.7).
Note that the experiment was run again 100 times for full 10 seconds for each planner on the third map to know the path lengths for full 10 seconds, since most of the runs plotted in fig. 4.8 did not last full 10 seconds.

Fig. B.2 shows the median path lengths of the planners over time for the third map.

Fig. B.3 shows the median time needed to find an initial path and the corresponding lengths.

Fig. B.4 shows the number of motions in the search graph at the end of the planning.

### B.2 Gap experiment

#### Setup

The gap experiment consists of six maps. Fig. B.5 shows details about the dimension of the maps used for the gap experiment.

#### Results

For the gap experiment, no information about the time needed to find an initial path and the corresponding lengths was collected. Also no information about the length of the paths for full 10 seconds was collected.

### B.3 Random map experiment

#### Results

In fig. B.6, the same results as shown in fig. 4.12 are shown including the 5th and 95th percentile.

Fig. B.7 shows the number of motions in the search graph at the end of the planning.

### B.4 Slalom experiment

#### Results

In fig. B.8 the same results as shown in fig. 4.15 are shown including the 5th and 95th percentile.

In fig. B.9, information about the time to find an initial path and the corresponding lengths are plotted. The same information can be seen in fig. B.8.

Fig. B.10 shows the number of motions in the search graph at the end of the planning.

### B.5 Mountain experiment

#### Results

In fig. B.11 the same results as shown in fig. 4.19 are shown including the 5th and 95th percentile.

Fig. B.12 shows the number of motions in the search graph at the end of the planning.
Figure B.2: Third map of the block experiment: Median length of the current best path over time for different planners. The error bars denote the 5th and 95th percentile. The second plot is a zoom-in on the first plot.
Figure B.3: Third map of the block experiment: The time needed to find an initial path and the lengths of the initial paths. The median is plotted in red, the 25th and 75th percentiles are plotted horizontally in blue and the 5th and 95th percentiles are plotted horizontally in black. The reason for the median time to find an initial path for CA-IRRT* being similar to DA-IRRT*, is that the range parameter for CA-IRRT* was set to \( \frac{\text{space extent}}{20} \) whereas for CA-IRRT*, D-IRRT* and D-RRT* it was set to \( \frac{\text{space extent}}{10} \). Hence, the tree of motions grows slower and denser and more time is needed to find an initial path.
Appendix B. More results

Figure B.4: 3rd map of the block experiment: The number of motions in the search graph after 10 seconds. The median is plotted in red, the 25th and 75th percentiles are plotted horizontally in blue and the 5th and 95th percentiles are plotted horizontally in black. D-RRT* does not use graph pruning and therefore has a lot more motions in the graph.

Figure B.5: Top view on the map for gap experiment. The gap width varies for each map from 5% (37.5m) to 20% (150m) in steps of 3% (22.5m). Recall that the airplane is assumed to have an extent of 10m by 10m by 10m. The height of the map is 1500m.
Figure B.6: Random map experiment: Median length of the current best path over time for different planners. The error bars denote the 5th and 95th percentile. The second plot is a zoom-in on the first plot with D-RRT* and D-FMT* hidden.
Figure B.7: Random experiment: The number of motions in the search graph after 10 seconds. The median is plotted in red, the 25th and 75th percentiles are plotted horizontally in blue and the 5th and 95th percentiles are plotted horizontally in black.

Figure B.8: Slalom experiment: Median length of the current best path over time for different planners. The error bars denote the 5th and 95th percentile. In more than 95% of all runs, D-RRT*, D-IRRT* and DA-IRRT* did not find a path (not plotted).
B.5. Mountain experiment

Figure B.9: Slalom experiment: The time needed to find an initial path and the lengths of the initial paths. The median is plotted in red, the 25th and 75th percentiles are plotted horizontally in blue and the 5th and 95th percentiles are plotted horizontally in black. D-RRT*, D-IRRT* and DA-IRRT* did not find a path in more than 95% of the runs, therefore, no data are plotted. In some runs, FMT* did not find an initial path with the initial batch of samples, but only in the subsequent batches when FMT* searched for an initial path with twice as much samples as in the initial batch of samples (alg. [12]). Therefore, some values for the time to find an initial path and the corresponding lengths are 0, as these values were initialized with 0. Note that the median and the other plotted statistical values are not influenced by these outliers.
Figure B.10: Slalom experiment: The number of motions in the search graph after 20 seconds. The median is plotted in red, the 25th and 75th percentiles are plotted horizontally in blue and the 5th and 95th percentiles are plotted horizontally in black.

Figure B.11: Mountain experiment: Median length of the current best path over time for different planners. The error bars denote the 5th and 95th percentile on the path length. This plot has to be studied together with fig. 4.20. Note that D-RRT*, D-IRRT* and DA-IRRT* need a long time to find an initial path. As a result, the path is close to optimal.
B.6 Kandervalley experiment

Results setup 1

In fig. B.13, the same results as shown in fig. 4.22 are shown including the 5th and 95th percentile.
In fig. B.14, information about the time to find an initial path and the corresponding lengths are plotted.
Fig. B.15 shows the number of motions in the search graph at the end of the planning.

Results setup 2, restricted height

In fig. B.16, the same results as shown in fig. 4.23 are shown including the 5th and 95th percentile.
In fig. B.17, information about the time to find an initial path and the corresponding lengths are plotted.
Fig. B.18 shows the number of motions in the search graph at the end of the planning.
Figure B.13: Kandervalley experiment: Median length of the current best path over time for different planners. The error bars denote the 5th and 95th percentile. Part of the first plot is shown in the second plot for better visibility.
Figure B.14: Kandervalley experiment: The time needed to find an initial path and the lengths of the initial paths. The median is plotted in red, the 25th and 75th percentiles are plotted horizontally in blue and the 5th and 95th percentiles are plotted horizontally in black. CA-IRRT* and CA-BIT* both find an initial path faster than any other path planning algorithm, to the disadvantage of longer path lengths.
Figure B.15: Kandervalley experiment: The number of motions in the search graph after 10 seconds. The median is plotted in red, the 25th and 75th percentiles are plotted horizontally in blue and the 5th and 95th percentiles are plotted horizontally in black.

Figure B.16: Kandervalley with restricted height experiment: Median length of the current best path over time for different planners. The error bars denote the 5th and 95th percentile.
Figure B.17: Kandervalley with restricted height experiment: The time needed to find an initial path and the lengths of the initial paths. The median is plotted in red, the 25th and 75th percentiles are plotted horizontally in blue and the 5th and 95th percentiles are plotted horizontally in black. CA-IRRT* and CA-BIT* both find an initial path faster than any other path planning algorithm, to the disadvantage of longer path lengths.
Figure B.18: Kandervalley with restricted height experiment: The number of motions in the search graph after 10 seconds. The median is plotted in red, the 25th and 75th percentiles are plotted horizontally in blue and the 5th and 95th percentiles are plotted horizontally in black.